

Amanda Oliver Calero

Julio Sala Gallardo

Gemma Sellés Lloret

Game Boy Learning Adventure

Amanda Oliver Calero

Julio Sala Gallardo

Gemma Sellés Lloret

Índice

Parte 1

Pueblo Game Boy

Bienvenida Pueblo Game Boy



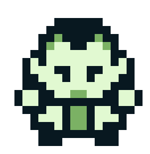
Profesor Retroman:

¡Hola! **Bienvenido** al mundo del desarrollo en Game Boy. Yo soy el **Profesor Retroman**.

Como tú, yo también empecé con curiosidad. ¿Cómo funcionan los juegos? ¿Cómo se programan esos gráficos tan simples y a la vez tan mágicos? Con el tiempo descubrí que, con el ensamblador, puedes controlar cada rincón de esta pequeña consola.

Ahora dime…

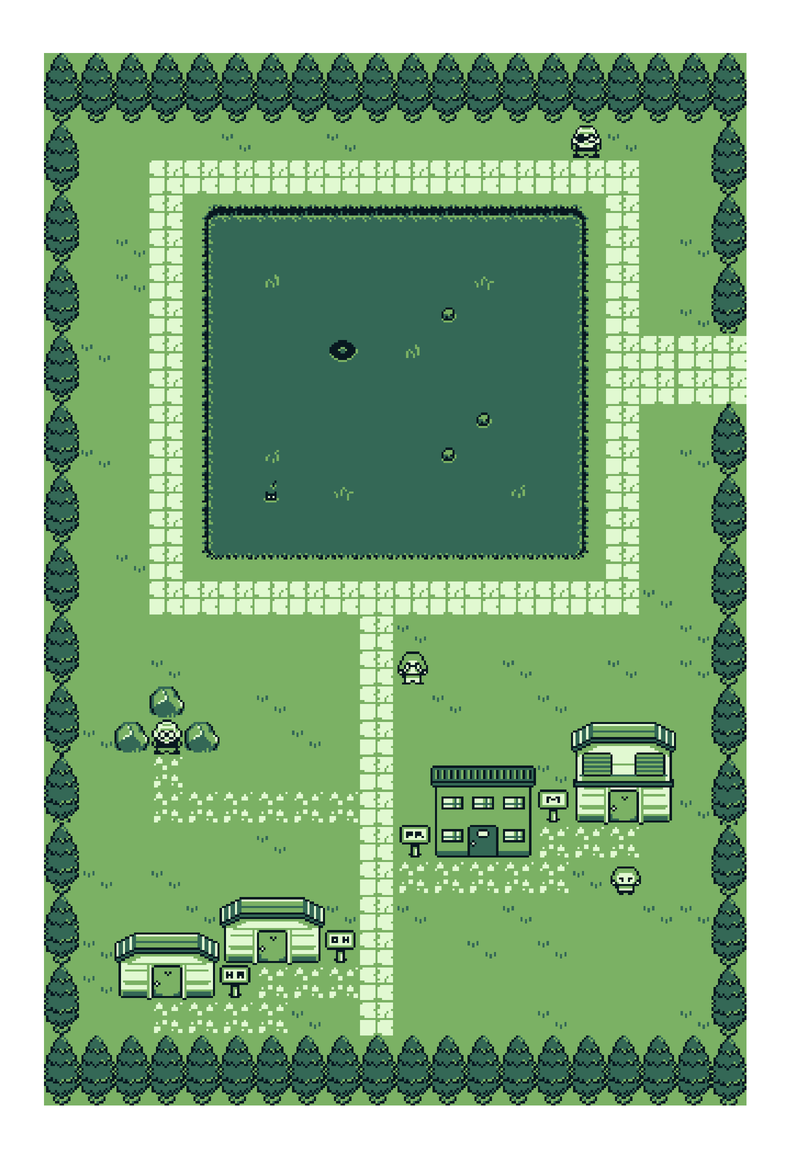
¿Estás listo para crear tu primer videojuego en Game Boy?



Mamá:

¿Dónde estás metido/a? La **comida** está lista desde hace rato y no pienso estar recalentando la comida como si esto fuera un bufé libre. Entra ahora mismo en casa que te he hecho tu plato favorito, y como no vengas en cinco minutos se lo come el perro, ¿me oyes?

Y por favor, cuando termines de comer no salgas volando otra vez con esos cascos puestos. Que tengo que darte unas cosas que me ha dado el Profesor Retroman para tí.



¿A dónde quieres ir?

- Dar una vuelta por el pueblo (página 13)
- Casa de tu Madre (página 14)
- Laboratorio del Profesor Retroman (página 18)
- Casa 0b1NAR10 y 0xH3XADEC1MAL (página 22)
- Taller de Hardwara (página 31)
- Ir a Pueblo Palette (página 55)

Pueblo Game Boy



Tina la niña:

Holaa, soy Tina ¿Sabes que? Ayer un juego me habló, me dijo: “It’s me…” pero no era Mario.



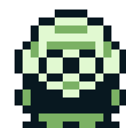
Chip el tuerto:

Hola niño/a, yo solía ser un programador como tú. Pero un día me hirieron en el ojo con un código mal ensamblado…



Doña Bitia:

Hola soy la bibliotecaria Bitia, este tomo habla de algo llamado… Z80. Suena a magia, ¿verdad?



Anciano Sospechoso:

Psst… oye, tú. Sí, el de cara soñadora y ojeras de tanto compilar. Me han dicho que tienes curiosidad por la **Game Boy**… que incluso andas jugueteando con ensamblador, qué valiente o insensato.

En cualquier caso, si de verdad quieres seguir ese camino, pásate por mis pruebas. Eso sí, asegúrate de ver antes al **Profesor Retroman** en su laboratorio, él te dará las bases para no acabar hablando con registros en tus sueños. Ven a visitarme en la página 36.

(Volver al mapa del pueblo, página 12)

Casa de tu Madre

Mamá:

Espero que te hayan gustado las lentejas. ¡Eh, eh, eh! ¿A dónde crees que vas jovencito/a? ¡Ni se te ocurra salir de esta casa sin tener **todo lo necesario** instalado! ¿Crees que puedes ir por ahí programando sin un buen entorno de desarrollo? ¡Ja! ¡**Vuelve aquí ahora mismo**!

Mira, te he dejado en la mesa todo lo que necesitas para desarrollar en Game Boy:

- **Ensamblador y Linker** (RGBDS), ¿cómo piensas hacer que la Game Boy te entienda?
- **Emuladores** (BGB y Emulicious), para que emules y depures tus juegos.
- **Editores de Tiles, Sprites y Tilemaps** (GBTD y GBMB), para que hagas tus assets.
- **Editor de Texto**, nada de escribir en papel como si estuviéramos en la prehistoria.

Cuando tengas todo instalado y funcionando, entonces, **y sólo entonces**, podrás salir a programar. ¡Y que no te pille diciendo que vas a "mirarlo luego", que te conozco!

Mamá:

Si quieres usar **linux** ves a la página 15 pero si prefieres usar una **máquina virtual** ya con todo configurado ves la página 16. Pero si ya tienes todo configurado ves a la página 17 que te quiero dar unos consejos de madre.

Instalación en Linux



Mamá:

Si quieres usar Linux, te recomiendo instalar **GBTelera**, que es un conjunto mínimo de utilidades para programar en ensamblador de **Game Boy**, inspirado en **CPCTelera**.

Se descarga desde https://github.com/lronaldo/gbtelera y se añade al path (instrucciones en github, no te hagas el despistado/a).

Estas son las herramientas que incluye:

- **RGDBS**: ensamblador y linker.
- **BGB / EMULICIOUS**: emuladores y depuradores de Game Boy.
- **GBMB / GBTD**: herramientas para diseñar mapas y tiles.
- **Makefile** personalizado: estructura de proyecto lista para compilar y generar ROMs.
- **Scripts personalizados**: para crear proyectos, ejecutar emuladores y descargar la DMG Boot ROM.

Requiere que tengas instalado **Wine** y **Java**.



Mamá:

Si ya lo tienes **todo instalado**, ves a la página 17 que te dé unos consejos de madre.

Uso de máquina virtual



Mamá:

A ver, criatura, si no quieres complicarte la vida instalando todo por separado, te tengo una solución fácil: **una máquina virtual con todo ya listo**. ¿Ves? ¡Yo sí que pienso en ti!

Esta **máquina virtual** viene con:

- **Linux Manjaro** como sistema operativo.
- **GBTelera** y CPCTelera.
- **Wine**, **Java** y compiladores de Rust y C++.

Y lo mejor de todo: **funciona con cualquier sistema operativo** con https://www.virtualbox.org/.

Se requiere tener una **CPU Intel**. Si la CPU es **ARM** (como en los últimos Mac) hay que hacer ajustes para la virtualización.

Aquí tienes el enlace para descargarla: https://archive.org/details/CPCtelera1.5-VM.



Mamá:

Si ya lo tienes **todo instalado**, ves a la página 17 que te dé unos consejos de madre.

Consejo de tu madre



Mamá:

Ay, mi niño/a… cómo has crecido. Parece que fue ayer cuando no sabías ni qué era un bit, y ahora mírate, listo/a para aventurarte en el mágico mundo del **ensamblador** de **Game Boy**.

Solo quiero que recuerdes unas cositas antes de salir por esa puerta:

- **Guarda tu código** a menudo y aprende a usar **Git**.
- Usa mucho el **depurador** y lee mucha documentación.
- **Ten paciencia**, que programar en Game Boy es como un puzzle con muchas piezas.

Y sobre todo, diviértete.

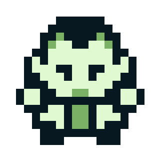
Así que, ¡adelante! El **mundo del ensamblador te espera**. Pero acuérdate de volver de vez en cuando, que aquí siempre habrá comida y un ordenador con todo instalado por si quieres volver a empezar de nuevo.

Acuérdate de visitar al **Profesor Retroman** y luego buscar a un **anciano** que está por el pueblo.

¡**Mucha suerte** en tu aventura, programador/a!

(Volver al mapa del pueblo, página 12)

Laboratorio del Profesor Retroman



Profesor Retroman:

Hola aprendiz, me presento: soy **Profesor Retroman**, uno de los pocos locos —digo, sabios— que aún conoce los secretos de la programación para **Game Boy**.

Llevo décadas entre bits, cartuchos y pantallas de fósforo verde. En mis tiempos mozos, me dedicaba al **Amstrad CPC**, programando en **ensamblador Z80**, y aunque esas épocas de cargar juegos desde cinta ya pasaron, las recuerdo con mucho cariño.

Ahora, mis circuitos están centrados en **Game Boy**. Este laboratorio es tu base de operaciones: yo te voy a enseñar lo básico para que puedas empezar a programar tus juegos desde cero.

Te voy a hacer una pequeña guía sobre cómo usar **GBTelera**.

Uso de GBTelera

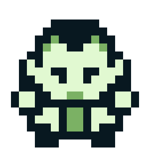


Profesor Retroman:

GBTelera se maneja desde la terminal, así que ya es hora de que aprendas a usarla, si no la tienes instalada acuérdate de volver a **Casa de tu Madre**, una vez instalada tendrás disponible estos comandos:

- **gbt_mkproject**: crea una carpeta con un nuevo proyecto base.
- **gbt_getDMGBoot**: descargar la BootROM original de Nintendo DMG-1 (*dmg_boot.bin*) y la copia en la carpeta de emuladores de GBTelera (/tools/emulators).
- **gbt_bgb**: lanza el emulador BGB con una ROM (.gb). Requiere WINE.
- **gbt_mkproject**: lanza emulicious con una ROM (.gb). Requiere Java.

Crear un proyecto con GBTelera



Profesor Retroman:

Para empezar, abre la terminal y en la carpeta donde quieras trabajar escribe:

```
gbt_mkproject <nombre-de-tu-proyecto>
```

Esto creará la carpeta “*nombre-de-tu-proyecto*”, con la estructura de subcarpetas y archivos de un proyecto de GBTelera. Luego entra en la carpeta y ejecuta, para ensamblar y generar la ROM game.gb:

```
make
```

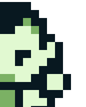
Para probarlo ejecuta, para lanzar los emuladores:

```
gbt_bgb game.gb

gbt_emulicious game.gb
```

Si tienes instalado WINE y Java, estos comandos lanzarán BGB o Emulicious y cargarán la ROM game.gb. Esta ROM no hace nada, por lo que simplemente deberías ver salir el logo de Nintendo y quedarse ahí parado.

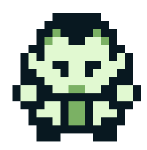
Importante en BGB



Profesor Retroman:

Si no te sale el logo de Nintendo, es porque no tienes puesto en el emulador de BGB la “*boot rom*”. En **BGB** hay que poner en “*System>Emulated System -> Gameboy; y en System>DMG bootrom –> la BootRom*” que descargamos antes, y poner el check a true de BootRoms enabled.

Estructura GBTelera



Profesor Retroman:

Una vez tienes un proyecto creado, tienes una carpeta con esto:

```
cfg/

Makefile

src/
```

Aquí una pequeña explicación de cada parte del proyecto:

- **cfg/**: incluye los ficheros de configuración completa del proyecto.
- **Makefile**: configuración del nombre, ensamblado y linkado.
- **src/**: contiene los archivos fuente del proyecto.

Dentro de **src/** tienes:

```
header.asm

main.asm
```

Para proyectos sencillos, podemos editar **main.asm** y escribir nuestro código debajo de la etiqueta global **main::**. Recuerda terminar con un bucle infinito o un **halt** (mejor si lleva **di** delante) para que el programa quede bloqueado.

El fichero **header.asm** tiene la información que la BootROM espera encontrar en los primeros 64 bytes (cabecera) de la ROM. Cosas como el nombre del juego, el desarrollador, el logo de Nintendo o qué recursos usa la ROM están ahí.

Si añades nuevos ficheros con extensión **.asm** en la carpeta **src/**, serán automáticamente ensamblados y enlazados. La configuración del proyecto está diseñada para hacer esto automáticamente.

Despedida de Profesor Retroman

Profesor Retroman:

Recuerda: cada gran aventura empieza con un **Make**, y una buena dosis de paciencia. No olvides volver a verme en mi Laboratorio cada vez que avances: cuando completes un proyecto, aprendas algo nuevo o simplemente quieras presumir de lo que has hecho.

Ahora que ya sabes como crear tus propios proyectos, acuérdate de buscar por el pueblo al **anciano de las pruebas** para continuar con tu aventura y aprender a dibujar tiles en pantalla.

Aquí estaré, con mis disquetes, mi soldador… y mis consejos retro.

¡Nos vemos en el próximo breakpoint!

(Volver al mapa del pueblo, página 12)

Casa 0b1NAR10 y 0xH3XADEC1MAL



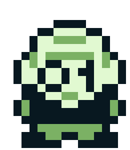
Abuelo Binario:

¡Ay niño/a, ven aquí! Déjame contarte cómo funciona esto del **binario**, el único sistema que vale realmente la pena. ¡Nada de esas tonterías de decimales, que tienen 10 números! ¡Con dos números, es más que suficiente hombre! Con **0** y **1**, el mundo es perfecto. Todo lo demás es solo para complicarse la vida. ¡Mira, te voy a contar una historia!

Yo, el **abuelo Binario**, nací en el glorioso mundo del **binario**, y a mis **1011111** años, te aseguro que no hay nada más sabroso que estos dos números, 0 y 1. ¡Qué gusto verlos bailar en las direcciones de memoria de la Gameboy!

¿Sabes qué es lo más bonito del binario? ¡Que todo tiene sentido! Nada de esos números locos como el **decimal** con 10 dígitos. ¡Qué barbaridad! Cuando yo era joven, veías un "9" y decías: "Pff, ¿para qué quiero un 9 si con 2 me basta?" El sistema binario tiene solo **2 números: 0 y 1**, cuando un bit está a 1, es como si el interruptor estuviera encendido; y cuando está a 0, pues apagado. ¡Así de simple! No hay necesidad de marearse con **10 números**. ¡Con dos bits ya se hace magia!

Convertir de Binario a Decimal



Abuelo Binario:

Entonces, a ver, imaginemos que tenemos el número "1011" (¡Oh, qué belleza, mira esos 1s y 0s!):

Binario	...	1	0	1	1
	2^{n+1}	2^3	2^2	2^1	2^0
Decimal	...	8	0	2	1

¡Ahora sumamos!

```
8 + 0 + 2 + 1 = 11
```

¡Ves! ¡Es una maravilla! Binario lo hace fácil, sin tener que andar con esos números gordos como el 7. **¡Más rápido que un cotilleo en radio patio!**

Aquí te dejo unos ejercicios para que practiques:

```
1010 → 10
1111 → 15
1111 1000 → 248
1001 0101 → 165
```

Convertir de Decimal a Binario

Abuelo Binario:

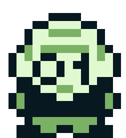
Ahora, te voy a contar cómo convertir un número decimal en binario. Sí, ya sé, no te pongas nervioso, es un viaje bonito y tranquilo. Vamos a tomar **13** (un número muy común, como la cantidad de veces que me he quejado del sistema decimal):

	Cálculo	Resto
Decimal	$13 / 2 = 6$	1
	$6 / 2 = 3$	0
	$3 / 2 = 1$	1
	$1 / 2 = 0$	1

Ahora, tomamos los restos de abajo hacia arriba: ¡**1101**!

Aquí te dejo unos ejercicios para que practiques:

```
3 → 11
20 → 10100
92 → 1011100
255 → 11111111
```



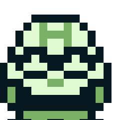
Abuelo Binario:

¿Sabes qué? Yo a los 1011111 años me sigo sintiendo joven. Y si algún día me ves escribiendo en **hexadecimal**, ¡sabrá Profesor Retroman que me habrá poseído un espíritu maligno!

¡Ah, pero espera un momento! Ahora que menciono el **hexadecimal**, te voy a dar una **gran noticia**… No creas que solo soy yo el único experto, también está mi hermano, el **cascarrabias del hexadecimal**. ¡Vaya personaje! No nos llevamos bien, ya sabes, los dos somos del mismo mundo, pero él… ¡él tiene una forma de pensar que no me gusta nada!

Mira, él es un **fanático** de esos números tan **largos y raros**. Ahora te va a decir que hables con él, porque según él, el **hexadecimal** es "más eficiente" o "más rápido para hacer ciertas cosas". ¡Ya te digo, es un **pelmazo**! Me tiene harto, siempre con su historia de que el **hexadecimal** hace todo más **compacto** o algo así.

Pero oye, **tú decides**. ¿Tienes ganas de aprender esos trucos complicados del hexadecimal? Adelante, si no siempre tienes el buen, simple y directo **binario**, ¡No hace falta más!



Abuelo Hexa:

¡Ah, sí, ahora me toca a mí! No te fíes de lo que te dijo ese **abuelo Binario**, no sabe lo que dice. Yo soy el **abuelo Hexa** y como soy gemelo del abuelo binario también tengo **5F** años, y sí, tengo un método **mucho más elegante** de hacer las cosas. Mi sistema de numeración **es el mejor**, y te voy a explicar por qué, aunque claro, esos del binario siempre me miran mal, no saben apreciar lo que de verdad importa.

Primero, déjame que te diga una cosa importante: el **hexadecimal** es como tener un **cajón con más compartimentos**, en vez de contar hasta 9 como esos raritos del decimal o hasta 2 como esos pobres del binario, nosotros llegamos hasta **F**. Es decir, en lugar de solo los números del 0 al 9, nosotros sumamos seis letras más: **A, B, C, D, E y F**. Así que **A vale 10**, **B vale 11**, **C vale 12**, **D vale 13**, **E vale 14** y **F vale 15**. ¡Mucho más eficiente!

¿Y sabes qué? Es mucho más **práctico**, si te quieres acordar de un número largo como **110101001001** en binario, va a ser un **rollazo**, pero si lo pasas a hexadecimal, te lo encuentras mucho más ordenado, como **D49**. ¡Mucho más fácil de leer!

Además si estás programando para **Game Boy**, te vendrá de perlas porque, para **representar números grandes**, el hexadecimal usa mucho menos espacio y es mucho más **compacto**.

Convertir de Binario a Hexadecimal

Abuelo Hexa:

¡Ay muchacho, esto es fácil! ¿Recuerdas cómo esos del binario sólo saben contar con **0 y 1**? Pues aquí entra el abuelo Hexa a poner orden. Vamos a convertir el número 101101110101, primero tienes que agrupar los bits de cuatro en **cuatro**:

Porque $2^4 = 16$, y 16 MI NÚMERO.

Luego, cada grupo lo conviertes a su valor hexadecimal:

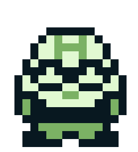
Binario	1011	0111	0101
Hexadecimal	B	7	5

Aquí te dejo unos ejercicios para que practiques:

```
01011001 → 59
110101 → 25
10111100 → BC
1111110 → 7E
```

Truco: si faltan para juntarlos de 4 en 4, añade 0s a la izquierda.

Convertir de Hexadecimal a Binario



Abuelo Hexa:

¡A ver, jovenzuelo! Cada número hexadecimal se **convierte en un grupito de cuatro bits**, porque sí, porque así lo dice el abuelo hexa y punto. No hay misterios, solo conviertes cada letra o número en su equivalente binario y los juntas.

Vamos convertir 2F3 a binario:

Hexadecimal	2	F	3
Binario	0010	1111	0011

Aquí te dejo unos ejercicios para que practiques:

```
A → 1010
F4 → 11110100
CF → 11001111
F0D → 111100001101
```

Convertir de Decimal a Hexadecimal

Abuelo Hexa:

Ay, el **decimal…** ese sistema arcaico que usan los que no han visto la luz. Para convertirlo a hexadecimal, tienes que dividir entre **16**, apuntar los residuos y luego leerlos al revés. Si te pierdes no vengas llorando.

Vamos a convertir 500 a **hexadecimal**:

	Cálculo	Resto
Decimal	$500 / 16 = 31$	4
	$31 / 16 = 1$	15
	$15 / 16 = 0$	1

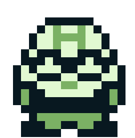
Ahora, tomamos los restos de abajo hacia arriba: ¡**1F4**!

Aquí te dejo unos ejercicios para que practiques:

```
235 → EB
94 → 5E
129 → 81
99 → 63
```

Si crees que dividir entre 16 es difícil, imagínate hacer esto con 8 bits en la cabeza mientras tu abuela te grita que la comida está lista.

Convertir de Hexadecimal a Decimal



Abuelo Hexa:

Aquí no hay trucos, solo matemáticas puras y duras. Cada dígito hexadecimal vale su peso en **potencias de 16**.

Vamos convertir **1C7** a **decimal**:

Hexadecimal	...	1	C (12 en decimal)	7
	$x * 16^{n+1}$	$1 * 16^{2}$	$12 * 16^{1}$	$7 * 16^{0}$
Decimal	...	256	192	7

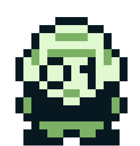
¡Ahora sumamos!

```
256 + 192 + 7 = 455
```

Aquí te dejo unos ejercicios para que practiques:

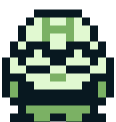
```
BA → 186
E1 → 225
16 → 22
67 → 103
```

¡Ahora ponte a practicar, que yo no tengo todo el día! Y si me haces otra pregunta sobre decimal, **te echo de mi casa**.



Abuelo Binario:

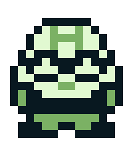
¡Bueno, chaval! Ya te hemos explicado **todo lo que necesitas saber** sobre el binario y el hexadecimal… ¡Ahora es tu turno de demostrar que no nos has hecho perder el tiempo!



Abuelo Hexa:

Necesitamos ayuda con un par de cosas de la casa para que pongas a prueba lo que has aprendido. Si nos ayudas, te ganas nuestro respeto… **si no, prepárate para un buen sermón de abuelos**.

La hora perdida en el reloj maldito



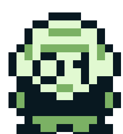
Abuelo Hexa:

Nuestro viejo reloj se ha **quedado parado**, pero como odiamos los números decimales, **la hora está en un formato distinto**:

- En la hora pone 1.
- En los minutos pone 2F.
- En los segundos pone 101000.

¡Averigua la **hora real** antes de que mi hermano lo arregle a martillazos (en formato decimal)!

El código de la caja fuerte



Abuelo Binario:

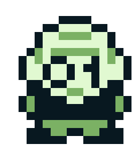
¡Ay muchacho, qué tragedia! Se nos ha **perdido el código de la caja fuerte** y ahora este viejo gruñón de mi hermano no para de decir que fue mi culpa. ¡Pero no es verdad! Él fue quien anotó el número en una servilleta mugrosa y ahora no sabemos dónde la dejó.

Menos mal que yo sí tengo cabeza, y me acuerdo de **algunas pistas**:

- En **decimal**, el número estaba entre **100 y 200**.
- En **binario**, me suena que se veía algo así: **1??? ???0**.
- En **hexadecimal**, mi hermano jura que empezaba por **A**.
- Ah, y me acuerdo que en **decimal**, la suma de los dígitos era **11**.

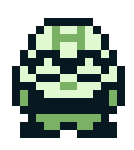
¡Rápido muchacho! Si no encontramos el número enseguida, mi hermano va a empezar a meter letras al azar, y ya sabes cómo acaba esto… **¡Con la caja bloqueada para siempre!**

Despedida de los Abuelos



Abuelo Binario:

Bueno chaval… Nunca pensé que alguien pudiera aprender tan rápido. Aunque todavía prefiero el binario, supongo que mi hermano Hexa no es tan inútil como pensaba…



Abuelo Hexa:

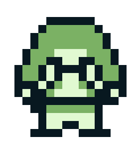
¡JA! ¿Ves? Al final mi sistema es el mejor. Pero oye, te defenderé si alguien te vuelve a hablar de decimal, porque eso sí que no lo soporto.

Los dos abuelos te miran con orgullo. Si has llegado hasta aquí oficialmente **eres parte de la familia de los números raros**. Ahora ve y usa tu conocimiento para programar tus jueguitos o simplemente impresionar a tus amigos.

Y recuerda… si alguna vez olvidas cómo convertir un número, **siempre puedes volver** a la vieja casa de los abuelos. Pero acuérdate de traer algo de comer la próxima vez, que aquí nadie vive del aire.

(Volver al mapa del pueblo, página 12)

Taller de Hardwara



Doña Hardwara:

¡Hola, querido! Soy **Doña Hardwara**, pasa, pasa, que te voy a contar una historia de esas que hacen latir más rápido el corazón de cualquier amante del hardware.

Resulta que allá por los **años 80**, un ingeniero japonés llamado **Gunpei Yokoi** y su equipo en **Nintendo** se propusieron crear una consola portátil que revolucionaría el mundo de los videojuegos: la **Game Boy**.

Hardware de la Game Boy

Doña Hardwara:

El cerebro de esta pequeña maravilla es el **Sharp LR35902**, un microprocesador de **8 bits** que combina lo mejor de las arquitecturas Zilog Z80 e Intel 8080. Este bichito corre a una velocidad de **4.19 MHz**, permitiendo ejecutar código ensamblador optimizado para las capacidades de la consola.

Mira, mira esta foto, cielo… ¿Ves esta preciosura? Es una **Game Boy desarmada**, pobrecita, con sus tripitas al aire. ¡Pero qué bonita es, incluso así! Aquí tienes su plaquita verde, con los circuitos bien marcaditos como si fueran venitas, ahí en el centro está su corazoncito: el **microprocesador**. ¡Ese que te conté antes, el Sharp LR35902!

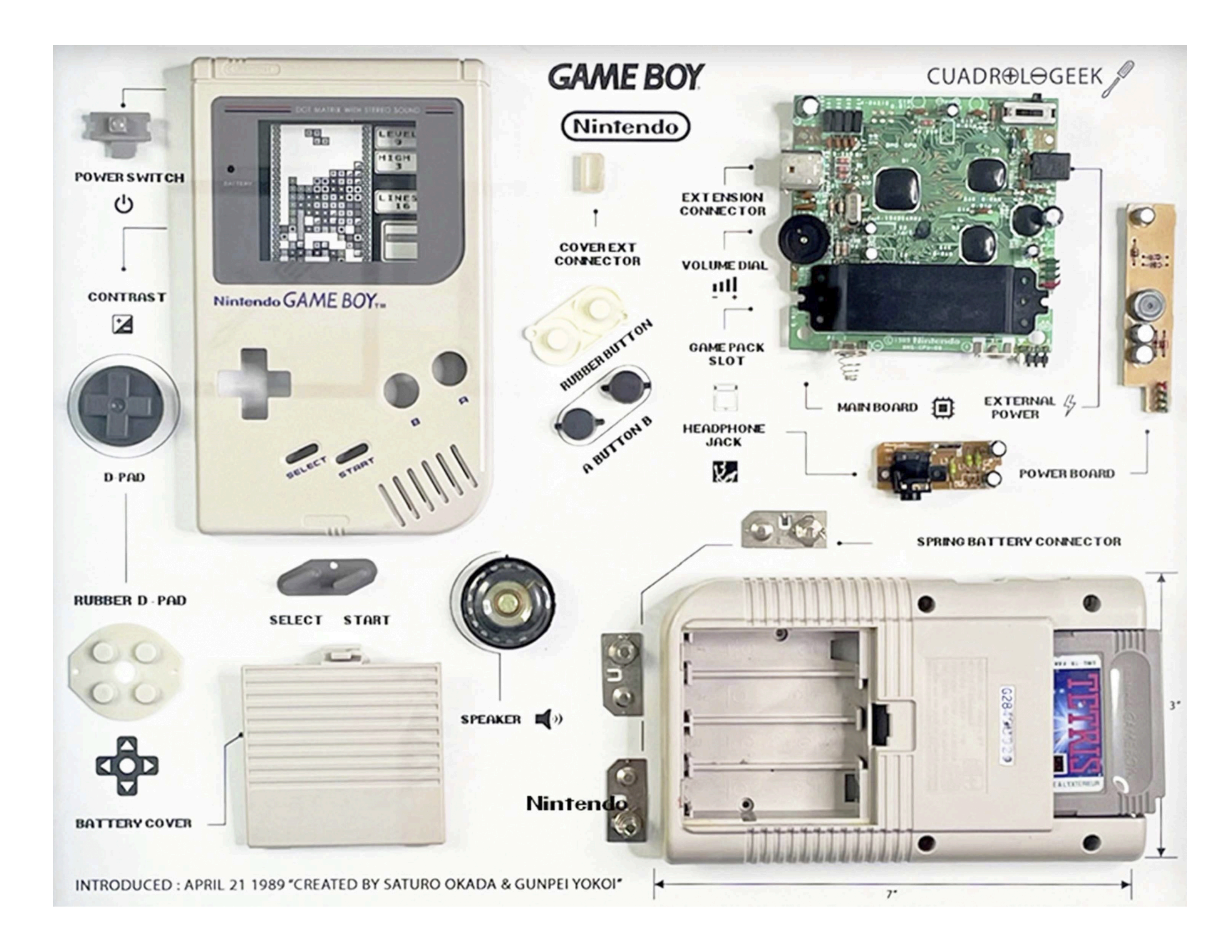


Foto de una Game Boy desarmada (cuadrologeek). Fuente: https://www.cuadrologeek.cl/cuadro-game-boy-classic

Pero eso no es todo, cariño… siéntate, que aún hay más que contarte. Mira, la **Game Boy** también venía con su **pantallita LCD** de unas **2.6 pulgadas**, no muy grande, pero suficiente para ver bien esos muñequitos moviéndose con gracia. Tenía una resolución de **160 x 144 píxeles**, y usaba cuatro tonos de verde, de esos que ya no se ven pero que tienen su encanto, ¿sabes?

En cuanto a memoria, traía **8KB de RAM** y otros **8KB de VRAM**, aunque los cartuchitos podían traer más si hacía falta. Y el sonido… ¡ay, el sonido! Tenía cuatro canales: dos de onda cuadrada, uno programables y uno de ruidito blanco, que hacía magia para esos efectos tan monos.

Y claro, todo eso lo movían con **cuatro pilas AA**. Sí, hijito, cuatro. Pero aguantan como campeonas, hasta **15 horas** dándote alegría sin parar, ¡que eso no lo hace ni mi tostadora nueva!

Y ahora, si me das un segundito, te explico cómo estaba organizada la memoria, que esto también tiene su aquel, ¿vale?

Mira:

- **ROM del cartucho:**
  - De 0000 a 3FFF → 16 Kb era fija, siempre ahí como el brasero del salón.
  - De 4000 a 7FFF → 16 Kb se podía cambiar si el cartucho tenía truco.
- **RAM:**
  - De 8000 a 9FFF → 8 Kb de VRAM, para los dibujitos.
  - De A000 a BFFF → 8 Kb de RAM extra del cartucho, si traía.
  - De C000 a CFFF → 4 KB de WRAM.
  - De D000 a DFFF → 4 KB de WRAM.
  - De FF80 a FFFE → 127 bytes de HRAM, poquita pero matona.
- **Otras cositas:**
  - De FE00 a FE9F → la OAM, donde se guardaban los sprites, esas figuras que se movían.
  - De FF00 a FF7F → los registros de entrada y salida… como las puertas del sistema, por decirlo así.

Ah, y ni se te ocurra meterte en la Echo RAM (de E000 a FDFF) o en la **memoria prohibida** (FEA0 a FEFF), que Nintendo dijo "¡**eso no se toca**!" y mejor no tentar a la suerte, ¿eh?

Evolución de las consolas de Nintendo

Doña Hardwara:

Y bueno, si miramos cómo han ido cambiando las consolas portátiles de **Nintendo**, te digo que esto ha sido como ver crecer a un nieto: empezó pequeñita, con su Game Boy del año 1989, y mírala ahora, convertida en toda una señorita moderna como la **Nintendo Switch**.

	Game Boy (1995)	Nintendo DS (2004)	Nintendo Switch (2017)
CPU	Sharp LR35902 (8 bits)	ARM946E-S (32 bits) ARM7TDMI (32 bits)	NVIDIA Tegra X1 (64 bits)
Frecuencia CPU	4,19 MHz	66 MHz 33 MHz	1,02 GHz (portátil) 1,78 GHz (dock)
RAM	8 Kb de RAM 16 Kb de WRAM 8 Kb de RAM externa	4 Mb de RAM 96 Kb de WRAM	4 Gb de RAM
ROM (cartucho)	32 Kb (ampliables)	hasta 64 Mb	hasta 64 Gb
Pantalla	1 pantalla LCD de 2,6”	2 pantallas LCD de 3”	1 pantalla LCD de 6,2”
Resolución	160 x 144 px	256 x 192 px	1280 x 720 px
Paleta de colores	4 tonos de verde	262,144 colores (18 bits)	16,7 millones de colores (24 bits)
Tarjeta de sonido	4 canales	16 canales	Puede emitir sonido envolvente 5,1 o 7,1
Batería	4 pilas AA (1,5V)	1000 mAh recargable (3,7V)	4350 mAh (16 Wh)
Duración media	10 - 15 horas	6 - 10 horas	2,5 - 6,5 horas
Controles	8 botones	12 botones	18 botones

Despedida de doña Hardwara

Doña Hardwara:

¡Vaya salto tecnológico, ¿verdad?! Pero te digo una cosa, cielo: por muy modernas que sean, ninguna de esas maquinitas nos va a quitar el cariño que le tenemos a nuestra viejita, la **Game Boy**, que tantas tardes nos dio diversión, con sus píxeles grandes y su corazón enorme.

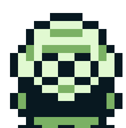
Y bueno, cielo, creo que con esto ya te he contado lo más importante. La Game Boy puede parecer sencilla, pero tiene su carácter, como una buena señora de su época. Así que si te vas a poner a programar en ella, te deseo toda la suerte del mundo.

Hazlo con mimo, **con paciencia**, y sobre todo **con amor**, que estas maquinitas lo sienten, ¿eh? Y si alguna vez te atascas, no te preocupes, respira hondo, tómate una infusión y vuelve a intentarlo. La Game Boy es dura, pero también agradecida.

¡**Mucho ánimo, mi amor**, y que tus bytes fluyan como el viento en una tarde de primavera pixelada!

(Volver al mapa del pueblo, página 12)

Las pruebas



Anciano de las Pruebas:

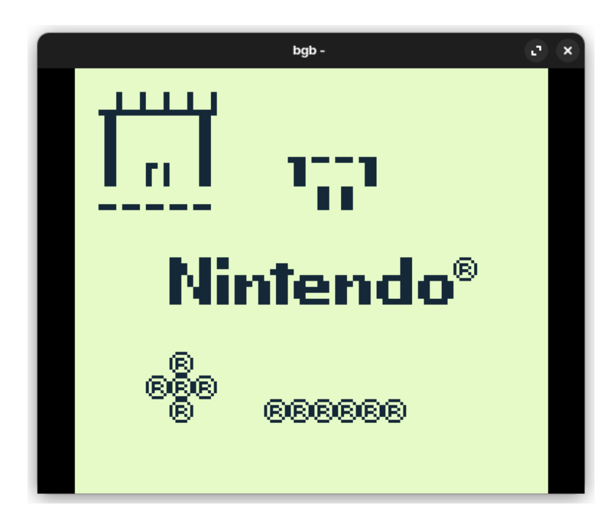
¡Eh, muchacho/a! Espero que hayas pasado a ver al **Profesor Retroman** antes de meterte en este lío, ¿eh? Si no lo has hecho, visítalo en su **Laboratorio** del Profesor Retroman, que ese buen hombre te enseñará cómo empezar con tus propios proyectitos para la Game Boy.

Ahora ven, siéntate aquí conmigo un momento y presta atención…

Yo soy el **Anciano de las Pruebas**, he oído que quieres crear tus propios videojuegos para la Game Boy… ¡y encima en **ensamblador**! Je, je, je… valiente eres, chaval, pero ya te aviso: no es tan fácil como encender la consola y ponerse a jugar.

Antes de salir del pueblo y lanzarte a esa gran aventura, tendrás que superar **unas pruebas** que te preparan bien para lo que se viene. ¿Y para qué sirven, te preguntarás? Pues para que aprendas a moverte como pez en el agua con el **debugger** de Game Boy y entiendas las **cosas básicas del ensamblador**. Nada de aporrear teclas sin ton ni son, ¿eh?

Si lo consigues… serás capaz de hacer dibujicos como estos:



Dibujar un Tile



Anciano de las Pruebas:

¡Hola joven! Te doy la bienvenida a las pruebas, en esta primera prueba vas a dibujar tu primer tile en la pantalla de la Game Boy. Pero antes de eso te voy a hacer una breve explicación de los registros, del formato de números que tienes que usar y por último dibujar un @ en la pantalla.

Explicación de los registros

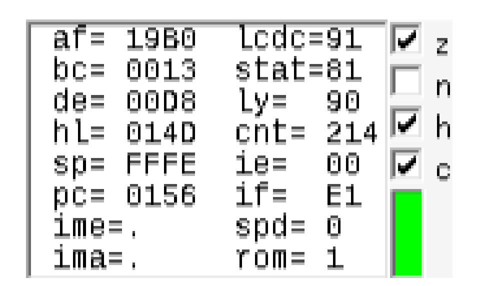


Anciano de las Pruebas:

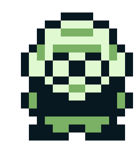
La **CPU** de nuestra querida Game Boy tiene **8 registros** con nombrecitos bien cortos: **a, b, c, d, e, f, h** y **l**. Estos registros son **pequeñas regiones de memoria con nombre** donde la CPU puede acceder de forma rápida y fácil. Como esos compartimentos secretos donde uno guarda caramelos para los nietos, jeje.

Ahora fíjate bien: algunos pares de registros de 8 bits se pueden combinar para formar registros más grandes, de **16 bits**. Hablamos de **af, bc, de** y **hl**. Pero ojo, estos registros combinados **comparten la misma región de memoria** que los registros de 8 bits. Eso quiere decir que si cambias el valor de uno de 16 bits, también cambias el valor de los de 8 bits que lo forman. ¡Y al revés también, claro!

Y todavía hay dos registros de 16 bits con funciones específicas: **pc** y **sp**, estos registros no tienen versiones de 8 bits.



Registros de 8 bits



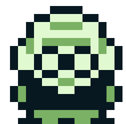
Anciano de las Pruebas:

El **registro a** se conoce como el **acumulador**. Es el registro más utilizado, casi todas las operaciones aritméticas y lógicas de 8 bits solo se pueden realizar en a.

Los **registros b, c, d, e, h,** y **l** son similares entre sí. Son menos versátiles que a y se usan principalmente para **almacenar datos temporalmente** mientras se trabaja con a.

El **registro f** es el registro de **flags** (banderas) y no se usa de forma directa. Contiene cuatro flags: **Z, N, H** y **C**. Los flags son básicamente **bits** que se activan y se desactivan según el estado de la última instrucción que las haya afectado. No todas las instrucciones afectan a todos los flags. Las que no se vean afectadas mantendrán su valor hasta que se ejecute una instrucción que las modifique. En **Pueblo Bandera, Flaguelito** te explicará más en detalle los flags.

Registros de 16 bits



Anciano de las Pruebas:

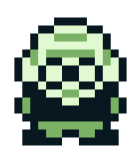
Los **registros bc, de** y **hl** se usan para almacenar valores de **16 bits** y para **direccionamiento indirecto**. Sin embargo, **hl** es más versátil, ya que también puede usarse para sumas de 16 bits y otras operaciones. Se puede considerar como el equivalente de 16 bits del acumulador a.

El **registro pc** también llamado **contador de programa**, apunta a la siguiente instrucción que la CPU va a ejecutar. Por ejemplo: supongamos que pc vale 100 y que la instrucción en la dirección 100 ocupa 3 bytes. La CPU buscará y ejecutará esa instrucción, y luego actualizará el

pc a 103 (100 + 3). Después ejecutará la instrucción en 103, luego actualizará el pc otra vez, y así sucesivamente.

El **registro sp** también llamado **puntero de pila** es el registro que apunta a la cima de la pila. La pila es una zona de memoria especial que se puede usar para guardar datos temporalmente, pero su principal uso es en llamadas a funciones.

Resumen de los registros



Anciano de las Pruebas:

A continuación una tabla con un resumen de los registros de la CPU de la Game Boy:

16 bits	Función
af	a: acumulador, f: flags/banderas (ZNHC)
bc	8/16 bits
de	8/16 bits
hl	8/16 bits, operaciones aritméticas
sp	stack pointer (puntero a la pila)
pc	program counter (contador de programa)



Formato de números

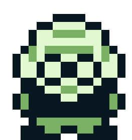
Anciano de las Pruebas:

Hay varios **formatos de números** definidos en el lenguaje de ensamblador. Los principales son: **hexadecimal,** binario y **decimal**. Las direcciones de memoria van a estar expresadas en hexadecimal, las flags en binario o en hexadecimal y el decimal se usará sobre todo en valores normales.

Los números hexadecimales llevan el prefijo **$ ($19)**, los números binarios llevan el prefijo **% (%00011001)** y los números decimales no llevan prefijo **(25)**.

Si tienes alguna duda sobre estos sistemas puedes visitar a mis viejos amigos.

Dibujado de un Tile



Anciano de las Pruebas:

La primera prueba es dibujar un **Tile** en pantalla, pero antes de nada, ¿Qué es un Tile?, un Tile es una **agrupación de píxeles** en forma de cuadrado de 8x8 píxeles.

Vamos a **dibujar** en la esquina superior izquierda la ® que hay al final del logo de Nintendo. Crea un nuevo proyecto y escribe en el apartado del main el siguiente código:

```
SECTION "Entry Point", ROM0[$150]

main::
    di                  ; Desactivar interrupciones
    ld       a, $19     ; Escribir en el registro A el $19
    ld [$9800], a       ; Escribir en la zona de memoria $9800 el valor de a
    jr @                ; Bucle infinito
```

Ensambla el código y comprueba el game.gb en el emulador que se ve la **® en la esquina superior izquierda**, si tienes algún problema de visualización o con el emulador el Profesor Retroman seguro que sabe como arreglarlo ves a su laboratorio.

Tienes que ver algo como esto:



Debugger / Depurador

Anciano de las Pruebas:

Vamos a continuar con las pruebas, en esta prueba vamos a entender parte por parte del **debugger** o depurador del emulador BGB. Te voy a hacer una pequeña introducción a las instrucciones en especial a las de **carga** y luego vamos a **experimentar** con el debugger.

Instrucciones

Anciano de las Pruebas:

Verás, muchacho, las **instrucciones** son las órdenes que entiende la **CPU de la Game Boy**. Cada una tiene un **mnemónico**, una palabreja corta que representa una acción concreta. Estas instrucciones están organizadas según lo que hacen, como si fueran herramientas bien colocadas en su sitio.

Échale un ojo a esta tabla:

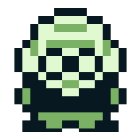
Categoría	Mnemónicos
Carga	ld, ldh
Saltos	jp, jr
Funciones	call, ret, reti, rst
Operaciones aritméticas	adc, add, dec, inc, cp, sbc, sub
Operaciones lógicas	and, or, xor
Operaciones de bits	bit, res, set, swap
Desplazamientos y rotaciones	rl, rla, rlc, rlca, rr, rra, rrc, rrca, sla, sra, srl

Operaciones de pila	pop, push
Otros	ccf, cpl, daa, di, ei, halt, nop, scf, stop

Si quieres ver la lista completa con todos los detalles, te dejo este enlace: https://meganesu.github.io/generate-gb-opcodes/

Link alternativo: https://web.archive.org/web/20250328101358/https:/meganesu.github.io/generate-gb-opcodes/.

Carga (ld y ldh)



Anciano de las Pruebas:

Ahora vamos a centrarnos en las instrucciones de **carga**, muchacho. **ld** y **ldh** sirven para **escribir valores** en registros o direcciones. Siempre hay un **destino** y una fuente, y se **escribe en el destino** el valor que hay en la fuente. No hay más misterio.

```
ld       a, $19      ; Escribir en el registro a el $19
ld       c, 57       ; Escribir en el registro c el 57 / $39
ld       a, [$9800]  ; Escribir en el registro a el valor contenido en $9800
ld    [hl], a        ; Escribir en la dirección contenida en hl el valor de A
```

¿Ves esos corchetes? Significa que hablamos de una **dirección de memoria**, no de un valor directo.

Es como usar **punteros en C++**:

- ld a, $19 → Escribe $19 en A.
- ld a, [$9800] → Escribe en A el valor que hay en la dirección $9800.
- ld [hl], a → Escribe el valor de A en la dirección que está en HL.

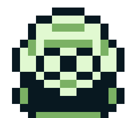
Usar un registro de 16 bits como dirección entre corchetes se llama **direccionamiento indirecto**.

Ten en cuenta que muchas instrucciones ld solo están disponibles para a o hl. Por ejemplo, cargar desde una dirección fija solo es posible con a. Igualmente, ciertas formas de direccionamiento indirecto sólo son posibles con hl, no con bc o de.

Ahora, hay una variante llamada **ldh**. Hace lo mismo, pero solo con direcciones entre **$FF00** y **$FFFF**, que es donde están los registros de entrada/salida, la HRAM e IE. ¿La ventaja? **Ocupa menos y es más rápido**.

```
ld  [$FF80], a   ; 3 bytes, 4 ciclos
ldh     [c], a   ; 1 byte, 2 ciclos (asumiendo que c contiene $80)
```

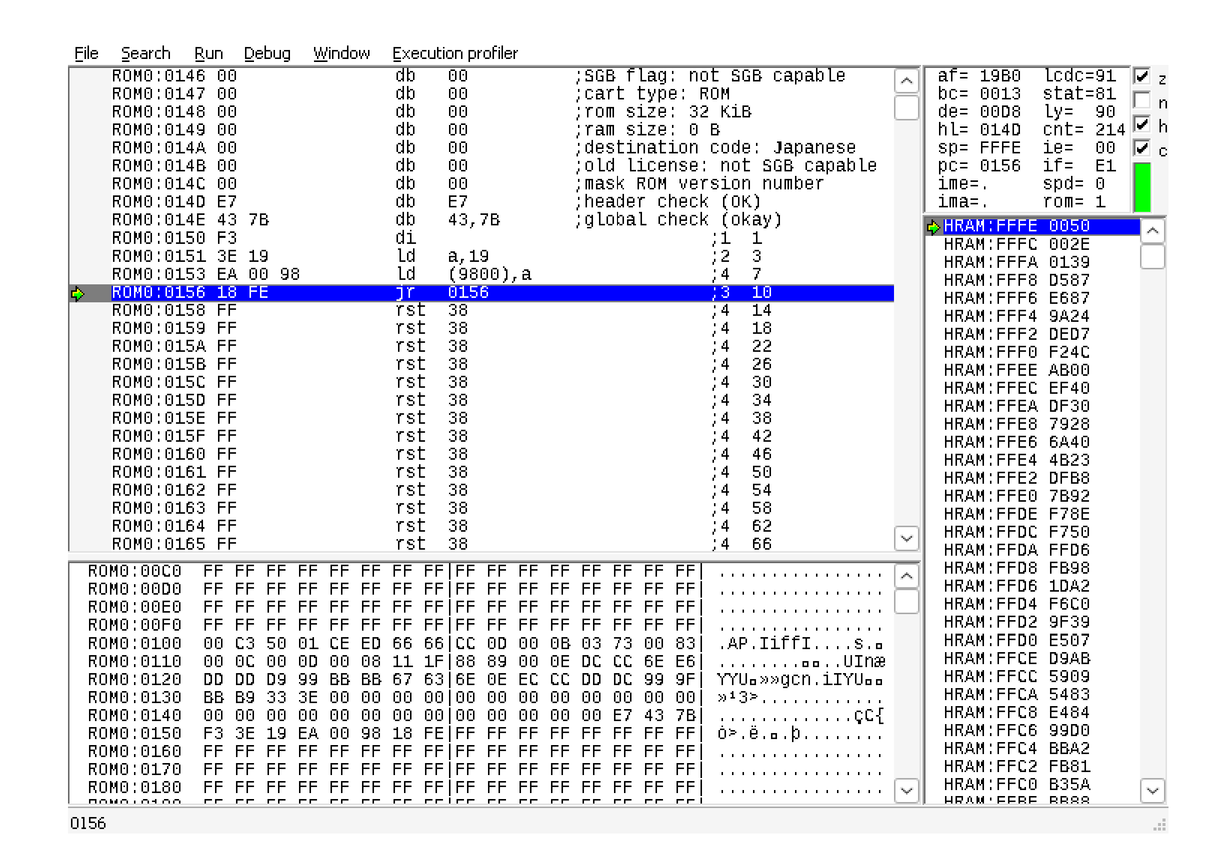
Debugger



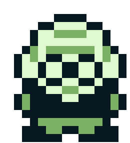
Anciano de las Pruebas:

Hay cuatro vistas en el debugger: **vista de código**, **vista de registros**, **vista de memoria** y **vista de la pila**. La vista de código, memoria y pila son diferentes representaciones del mapa de memoria. La vista de registros es una forma rápida y útil de consultar el valor de los registros de la CPU, los flags de estado y algunos registros importantes de E/S.

Este es el estado del debugger al terminar la prueba anterior:



En cualquier momento durante la ejecución de un programa en BGB, puedes pulsar la tecla **ESC** para abrir el debugger. Esto **pausa la ejecución** del programa y muestra el debugger.



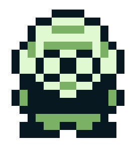
Anciano de las Pruebas:

La **vista de código** presenta el mapa de memoria como un conjunto de instrucciones.

Es posible avanzar paso a paso en el código y colocar puntos de ruptura (breakpoints) en esta vista. Es donde los programadores suelen pasar la mayor parte del tiempo al hacer la **depuración**. El formato de cada línea en esta vista es el siguiente:

memory :	address	bytes	disassembly	; comment	cumulative cycles
ROM0 :	0156	18 FE	jr 0156	; 3	10

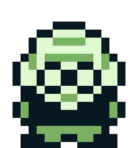
```
ROM0:0146 00          db   00         ;SGB flag: not SGB capable
ROM0:0147 00          db   00         ;cart type: ROM
ROM0:0148 00          db   00         ;rom size: 32 KiB
ROM0:0149 00          db   00         ;ram size: 0 B
ROM0:014A 00          db   00         ;destination code: Japanese
ROM0:014B 00          db   00         ;old license: not SGB capable
ROM0:014C 00          db   00         ;mask ROM version number
ROM0:014D E7          db   E7         ;header check (OK)
ROM0:014E 43 7B       db   43,7B      ;global check (okay)
ROM0:0150 F3          di                          ;1   1
ROM0:0151 3E 19       ld   a,19                   ;2   3
ROM0:0153 EA 00 98    ld   (9800),a               ;4   7
ROM0:0156 18 FE       jr   0156                   ;3   10
ROM0:0158 FF          rst  38                     ;4   14
ROM0:0159 FF          rst  38                     ;4   18
ROM0:015A FF          rst  38                     ;4   22
ROM0:015B FF          rst  38                     ;4   26
ROM0:015C FF          rst  38                     ;4   30
ROM0:015D FF          rst  38                     ;4   34
ROM0:015E FF          rst  38                     ;4   38
ROM0:015F FF          rst  38                     ;4   42
ROM0:0160 FF          rst  38                     ;4   46
ROM0:0161 FF          rst  38                     ;4   50
ROM0:0162 FF          rst  38                     ;4   54
ROM0:0163 FF          rst  38                     ;4   58
ROM0:0164 FF          rst  38                     ;4   62
ROM0:0165 FF          rst  38                     ;4   66
```



Anciano de las Pruebas:

La **vista de registros** muestra el valor de varios registros. Algunos de estos registros te deberían resultar familiares, como los registros de la **CPU**.

```
af= 19B0   lcdc=91   z
bc= 0013   stat=81   n
de= 00D8   ly=   90  h
hl= 014D   cnt= 214  c
sp= FFFE   ie=   00
pc= 0156   if=   E1
ime=.      spd=  0
ima=.      rom=  1
```



Anciano de las Pruebas:

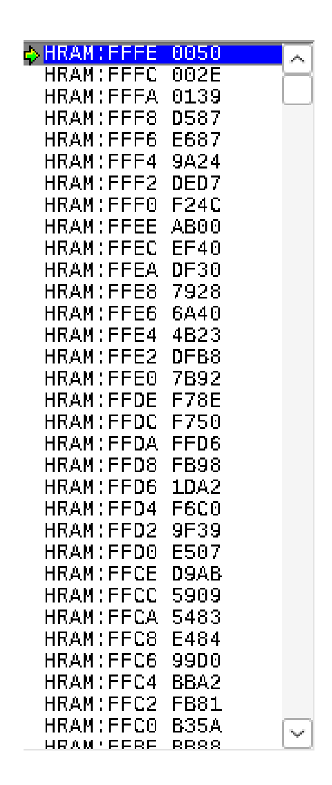
La **vista de memoria** muestra el **mapa de memoria** como una **serie de bytes.**

Hay **16 bytes** por línea, así que una sola pantalla representa una buena cantidad de datos. Esta vista se usa típicamente para comprobar que la memoria contiene los valores esperados en las direcciones esperadas. También te permite modificar cualquier byte de la memoria, lo cual es muy útil para probar o reproducir escenarios raros.

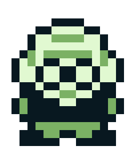
```
ROM0:00C0  FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF| ................
ROM0:00D0  FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF| ................
ROM0:00E0  FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF| ................
ROM0:00F0  FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF| ................
ROM0:0100  00 C3 50 01 CE ED 66 66|CC 0D 00 0B 03 73 00 83| .AP.IiffI....s.▫
ROM0:0110  00 0C 00 0D 00 08 11 1F|88 89 00 0E DC CC 6E E6| .........▫▫..UInæ
ROM0:0120  DD DD D9 99 BB BB 67 63|6E 0E EC CC DD DC 99 9F| YYU▫»»gcn.iIYU▫▫
ROM0:0130  BB B9 33 3E 00 00 00 00|00 00 00 00 00 00 00 00| »¹3>............
ROM0:0140  00 00 00 00 00 00 00 00|00 00 00 00 00 E7 43 7B| .............çC{
ROM0:0150  F3 3E 19 EA 00 98 18 FE|FF FF FF FF FF FF FF FF| ó>.ê.▫.þ........
ROM0:0160  FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF| ................
ROM0:0170  FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF| ................
ROM0:0180  FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF| ................
```

Anciano de las Pruebas:

La **vista de la pila** está dedicada principalmente a la pila. Aunque muestra todo el mapa de memoria, igual que la vista de datos, tiene varias características pensadas para la inspección de la pila.



Experimenta



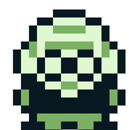
Anciano de las Pruebas:

¡Ay, muchacho, ahora que ya has visto cómo va ese cacharro del debugger, es tu turno de juguetear un poco con él!

- Prueba a **cambiar el valor del PC**, que es como decirle a la consola “oye, empieza desde aquí en vez de allí”. Verás cómo el programa se comporta distinto, ¡como si lo despistaras!
- Mételes mano a los **datos del mapa de memoria**, cámbialos a ver qué pasa. A veces un simple numerito lo cambia todo, como cuando le echas sal al café sin querer.
- Y no te olvides de los **breakpoints**. Ponlos donde te dé la gana, que sirven para parar el programa justo cuando tú quieras, como si le dijeras “¡Quieto ahí, chaval!”

No tengas miedo de liarla. Si algo se rompe, mejor. Así aprendes. Aquí no pasa nada por meter la pata. ¡Eso sí, no le eches la culpa al abuelo si explota todo!

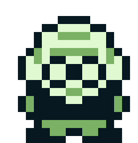
Figuras



Anciano de las Pruebas:

Ahora que ya dominas un poco el debugger y sabes cómo cargar valores en memoria, vamos a **dibujar nuestras primeras figuras** en la pantalla de la Game Boy. Lo haremos directamente en el BG Map, que empieza en la dirección $9800.

Pantalla y VRAM



Anciano de las Pruebas:

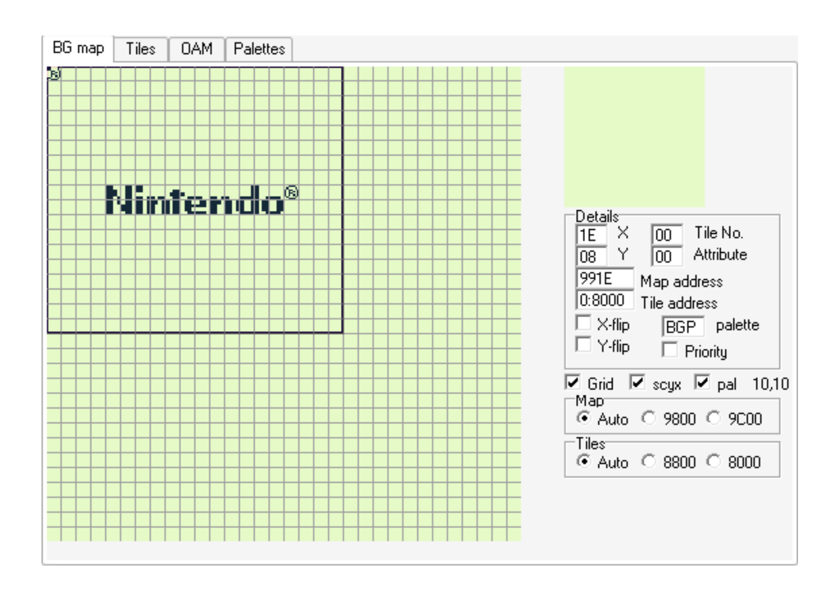
Vamos a recordar el código de la primera prueba:

```
ld        a, $19
ld  [$9800], a
```

Esto NO se puede hacer, porque no se puede escribir directamente un valor en memoria.

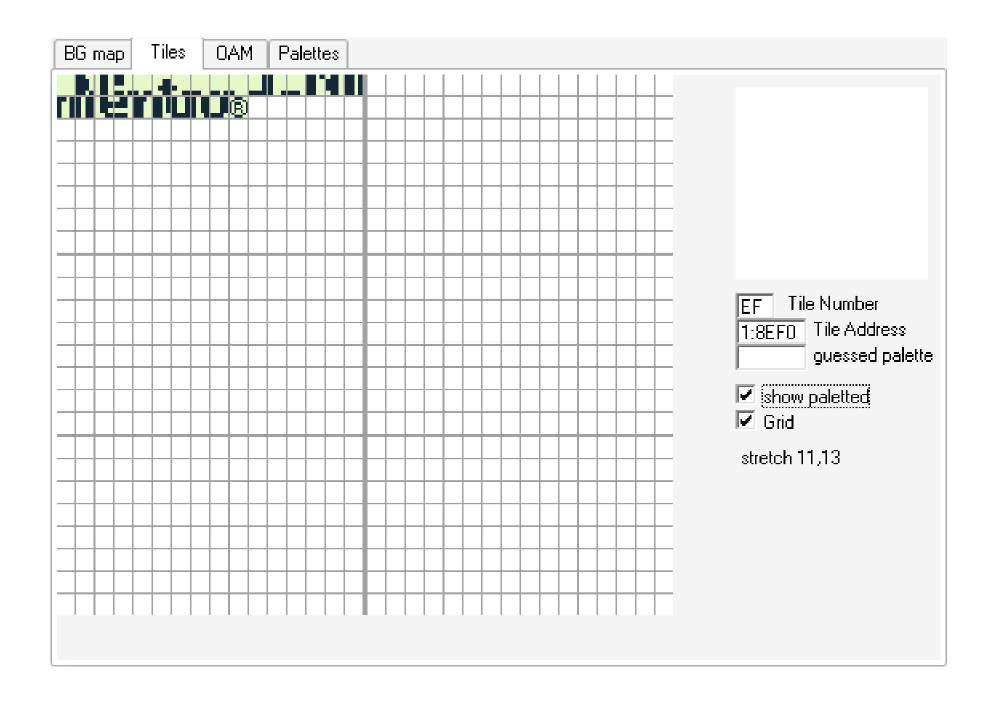
```
ld  [$9800], a
```

El **BG Map** (Background Map) es una parte de la memoria de video (VRAM) que indica qué tiles se dibujan en pantalla y dónde. Cada celda del BG Map contiene un índice de tile, que apunta a un gráfico en la memoria de tiles. La Game Boy usa 32x32 tiles para el fondo (pantalla completa más scroll), y el BG Map empieza en la dirección **$9800**, según la configuración.



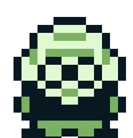
Con este código estamos dibujando un tile dentro de la VRAM, que se puede visualizar en el VRAM Viewer del emulador de BGB. En concreto, el tile número **$19**.

¿Pero qué es $19? Eso es un **índice de tile** dentro de la VRAM, que puedes visualizar en el **VRAM Viewer** del emulador. Cada tile tiene su numerito, y al escribir ese número en una dirección del mapa ($9800 en adelante), la consola lo pinta en pantalla.



En el **Pueblo Palette**, **Paquito el de la VRAM** te lo explicará más en detalle, no sin antes intentar que alquiles una película de los 80s.

Dibuja tu figura



Anciano de las Pruebas:

A ver si te animas a dibujar una figura sencilla en pantalla.

Haz tus pruebas, inventa una **letra**, un **muñequito**, lo que quieras. No hace falta que se vea bonito: ¡lo importante es que entiendas cómo funciona!

Cuadrado

Un cuadrado 2x2 en la esquina superior izquierda:

```
ld        a, $19
ld  [$9800], a
ld  [$9801], a
ld  [$9820], a
ld  [$9821], a
```

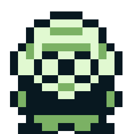
Recuerda que cada fila en el BG map son 32 tiles, así que para ir una fila abajo hay que sumar $20 a la dirección.

Triángulo

Una escalera en la esquina superior izquierda:

```
ld        a, $19
ld  [$9801], a
ld  [$9820], a
ld  [$9821], a
ld  [$9840], a
ld  [$9841], a
```

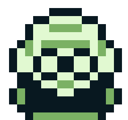
Saltos



Anciano de las Pruebas:

Vamos con la **última prueba** antes de la gran final. Hoy te voy a enseñar cómo hacer **comentarios** en tu código y cómo usar los **saltos** de forma básica, que los hay de varios tipos, no te vayas a liar.

Comentarios



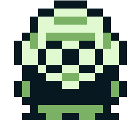
Anciano de las Pruebas:

Mira, todo lo que pongas después de un "**punto y coma**" (;) se considera **comentario**. El ensamblador lo **ignora totalmente**, como cuando me hablas mientras veo mi novela.. Es equivalente a usar // en Java, C o C++ o usar # en Python.

Y escucha bien: los comentarios son **tu salvavidas mental**. El código ensamblador es tan explícito y detallado que hasta lo más simple parece un **TFG**. Comenta lo que haces y por qué lo haces. No seas vago. Te lo agradecerás tú mismo y cualquiera que lea tu código más adelante.

Ah, y ni se te ocurra buscar los comentarios en el debugger… ¡porque no salen!

Saltos



Anciano de las Pruebas:

Ahora vamos con los saltos es una **forma de cambiar** el valor del **registro pc**, te ayudan a tomar control del flujo de la ejecución de las instrucciones y son fundamentales para **condicionales**, **bucles** o incluso para hacer **funciones**.

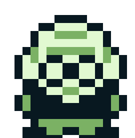
Saltos Incondicionales vs Condicionales



Anciano de las Pruebas:

Hay dos tipos de saltos: **incondicionales** y **condicionales**. Un **salto incondicional** solo toma como parámetro una dirección a la que saltar, y siempre actualiza el valor de pc con esa dirección. Un **salto condicional** toma dos parámetros: una condición y una dirección, solo actualiza el valor del pc si cumple la condición.

Saltos Absolutos vs Relativos



Anciano de las Pruebas:

Hasta ahora solo hemos hablado de los tipos de saltos condicionales y los incondicionales, pero estos se aplican a los dos tipos de saltos que se pueden hacer.

El código máquina **jr es un salto relativo**, toma como parámetro una dirección de 16 bits y la convierte en una de 8 bits con signo. Este salto/desplazamiento se usa para saltar relativamente desde la dirección justo después del salto.

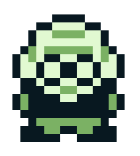
La instrucción **jp es un salto absoluto**. Toma una dirección de 16 bits que se usa tal cual en el código máquina.

¿Te suena esto?

```
jr @
```

El @ **representa la dirección actual**. Así que el programa se salta a sí mismo una y otra vez. Un **bucle infinito** de toda la vida, ideal para dejar el programa esperando al final.

Etiquetas



Anciano de las Pruebas:

Y por cierto, las **etiquetas** como **main** o **etiqueta** no son más que formas de ponerle nombre a una dirección del programa. Así luego puedes hacer **jp** o jr a esa parte del código sin tener que contar bytes como un loco.

```
main::

    jp etiqueta

    ld a, b

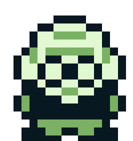
    etiqueta:

    ; jp saltará aquí

    jr @
```

Si esto es algo parecido a un **bucle**, en **Pueblo Bandera**, en la **noria de Loopita**, se te explica más en profundidad los bucles.

¿Funciones?



Anciano de las Pruebas:

Bueno bueno, a ver si te atreves con este **mini reto** que te tengo preparado.

Quiero que montes un sistema con saltos como si hicieras una **función a mano**. Imagina que estás en tu código principal de dibujar algo en pantalla, separa esa parte del código y ponla en otra parte. Pues quiero que uses un **salto** para ir a esa parte del código donde haces el pintado, y cuando termines, que **vuelvas justo al sitio de antes** y sigas con la ejecución normal.

Y nada de usar funciones ni otras instrucciones chulas, ¡a la antigua, solo con **jp, jr y etiquetas**!

Prueba Final

Anciano de las Pruebas:

Ya has aprendido a escribir valores en memoria, a usar la instrucción **ld**, a hacer saltos con **jp** y **jr**, y a explorar cómo funciona la pantalla de la Game Boy a través del **BG Map**. Es hora de juntar todo ese conocimiento en un ejercicio libre: crear tu propio dibujo en la pantalla.

Empieza con una **idea sencilla**: un cuadrado, una cruz, una letra, un símbolo… lo que tú quieras. Usa el BG Map como tu lienzo y los tiles como pinceles. Puedes observar cómo cambian los tiles en el VRAM Viewer del BGB y probar diferentes valores para ver qué aparece. No importa si no sale perfecto a la primera, lo importante es experimentar y entender cómo tus instrucciones afectan lo que se ve en pantalla.

Este reto no tiene una solución correcta, lo que cuenta es que te animes a probar, a equivocarte, y a aprender jugando. **Así es como se aprende de verdad**.

(Volver al mapa del pueblo, página 12)

Parte 2

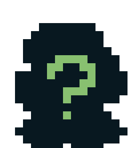


Pueblo Palette

Bienvenida Pueblo Palette

Después de tus primeros pasos con Game Boy, llegas al siguiente pueblo. Desde la entrada se ve en el centro un magnífico edificio rodeado de un precioso jardín. Los ciudadanos disfrutan del buen tiempo y las vistas alegremente. Se les ve bohemios; hay quien pinta por los alrededores.

Te acercas embelesado por el encanto de la escena, ¡hace un día de postal! Cuando de repente te tropiezas con...



Misterioso pelirrojo:

¡Eh, tú! Ve con más cuidado. No te he escuchado acercarte... ¿No eres de aquí, verdad? Tus pintas te delatan un poco, ja, ja, ja. ¿Qué estás, perdido por el Pueblo? Si quieres, te puedo hacer un tour :) ¡Acompáñame!
¡Oh! ¿Pero dónde están mis modales? ¡Aún no me he presentado! Mi nombre es Vincent, pero la gente me suele llamar por mi apellido, así que me puedes llamar **V. Blank Gogh**.



V. Blank Gogh:

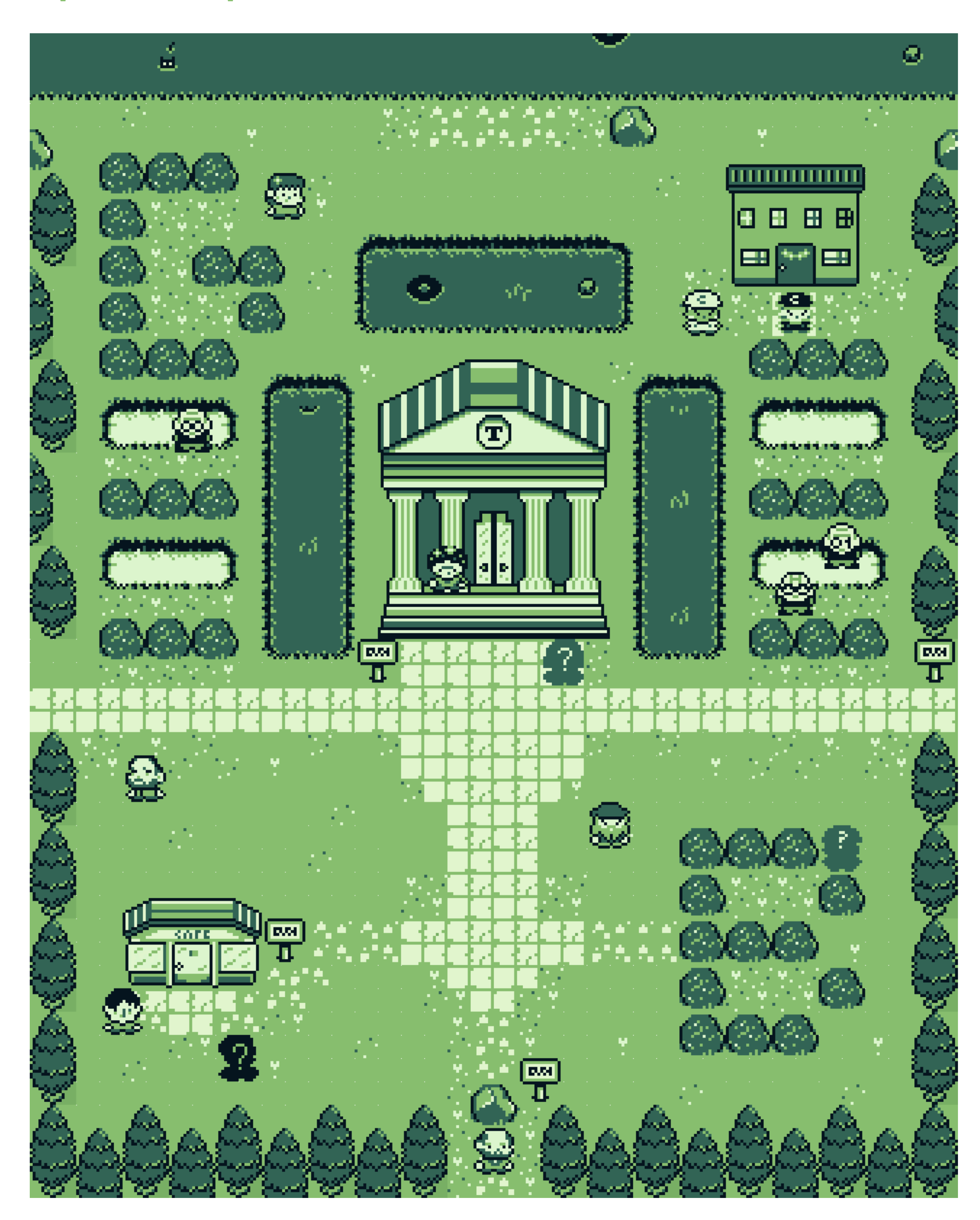
Ahora sí que sí, ¡permíteme que te muestre todo esto!



V. Blank Gogh:

Bien, veamos por dónde empezamos…

Mapa del pueblo Palette



- **Jardín del museo**: aaah... Los jardines... Llenos de vida y de bohemios... Fue lo que más eché en falta en mis años de internado... Si quieres ir a ojear qué hacen los **artistas**, seguro que estarán encantados de mostrarte su trabajo. Aunque te recomiendo que tengas cuidado, por los jardines a veces merodean los **hermanos Palette**... Esos diablillos...
- **Museo de *tiles***: la joya de la corona de nuestro pueblo, ¡allí se encuentra una de las mayores exposiciones de *tiles* del mundo! También encontrarás todo lo que necesitas saber sobre ellos. La **Directora Pombalina** seguro que te puede hacer una visita guiada y explicarte lo que necesites. Puede que encuentres algún artista en alguna sala.
- **Cafetería del museo**: hacen unos bocatas espectaculares. Siempre que quieras hablar conmigo, me puedes encontrar ahí junto a **Paquito** o a **Blanktisse**.
- **Casa de los hermanos Palette**: aquí es donde viven los jóvenes **hermanos Palette**; a veces los puedes encontrar aquí ideando nuevas travesuras. Serán cuatrillizos, pero cada uno es único. A los únicos que temen es a los abuelos. ¡Ellos sí que saben ponerlos en su sitio!

Creo que ya te he explicado lo más relevante. Si me disculpas, es hora ya de almorzar. Voy camino de la cafetería. Si tienes hambre, me puedes acompañar.



V. Blank Gogh:

Te recomiendo, si no sabes nada de *tiles* ni de la VRAM ni de los principios **básicos de gráficos**, que hables con tanta gente como te sea posible.

¡Y casi se me olvida! En el **pueblo de al lado** tienen montada una feria espectacular (si no la has visto ya), ¡no te la puedes perder! Solo tienes que acercarte a la estación **Cable Link** para llegar en un santiamén. ¡Nos vemos!

¿Dónde quieres ir?

El texto **enfatizado** indica que es un reto. El poblado se puede leer en el orden en el que está por defecto o saltando entre las zonas. Incluso en una misma zona se puede saltar entre apartados.

Cafetería del Museo

Lugar de descanso y reunión para los artistas de Pueblo Palette. De aquí han surgido buenas ideas para mejorar su técnica y estilo. Hacen unos bocatas de lomo espectaculares.

V. Blank Gogh:

Pero mira a quién tenemos aquí, ¡si es nuestro nuevo pequeño artista! Porque has venido a la ciudad para convertirte en uno, ¿verdad? Si has venido por la comida, no te juzgo... Anda, prueba un poco...

Ahora con el estómago lleno ya podemos empezar. Un buen artista debe conocer sobre qué superficie trabaja, qué **lienzo** vamos a usar. Ya veo que traes contigo la Game Boy; algo podremos hacer con ella.

Pantalla de la Game Boy



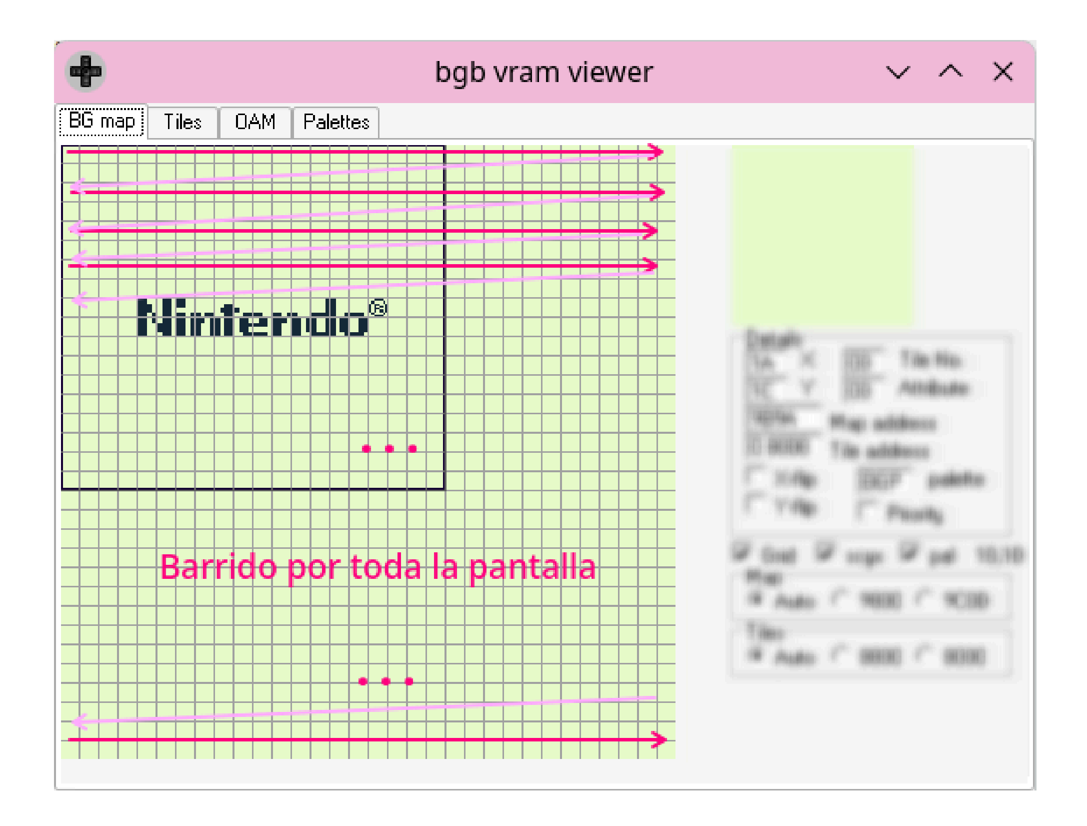
V. Blank Gogh:

Nuestro lienzo será la pantalla: para ello debemos saber algunas nociones antes de ponernos a dibujar.

La Game Boy tiene una pantalla **LCD**, *Liquid-Crystal Display*, de 160×144 píxeles. Creo que viniendo de Pueblo Inicial ya te habrán comentado algo. Los píxeles no se manipulan individualmente, sino que se agrupan en grupos de 8×8 píxeles llamados *tiles*. De no ser así, sería muy costoso para la CPU.

Los *tiles* que se muestran en pantalla se encuentran en una región concreta de la memoria de la consola, la **VRAM**, *Video RAM*. La encargada de dibujarlos por pantalla es la **PPU**, *Pixel Processing Unit*.

Para dibujar un *frame*, la PPU traza 154 *scanlines*, que son las líneas de barrido que traza la pantalla para pintar filas de píxeles. A lo mejor eres muy joven, pero ¿has visto alguna vez alguna tele retro? Esas que eran muy pesadas y grandotas... pues tenían un funcionamiento similar. Este *barrido* de pantalla pinta las filas de arriba a abajo y cada fila de izquierda a derecha.



Ahora más adelante te contaré cómo destripar la pantalla con BGB como en la imagen que te acabo de enseñar.

De la manera en que pinta la PPU es como está ordenada la zona de la pantalla en la VRAM. Saber esto, en un futuro, cuando pases por el **campamento arte**, puede que te enseñen algunos efectillos chulos...

Tiles y tilemap

Los *tiles* son como si fueran ingredientes de una receta culinaria. Dispones de una serie de ellos y los puedes combinar para las distintas elaboraciones del plato, p. ej., *tu personaje, una casa, un árbol, un superbocata de lomo*... Creo que me he quedado con hambre.

Si los *tiles* son ingredientes, el *tilemap* es una despensa infinita de ellos. Conforme los necesites para tu receta, simplemente referencia al que vas a usar.

V. Blank Gogh:

Tanto hablar contigo me ha dado sed, *no como los 3 bocatas que me he metido entre pecho y espalda*... ¡Eh! ¡Paquito! No te vayas tan deprisa, tengo una peli que devolverte, que después metes unos sablazos con las multas que da gusto.

Paquito:

¿Justo ahora me paras? Sí he estado todo el rato aquí, que tengo que volver al videoclub, macho... Luego vuelvo y tengo a todos los vecinos preguntando por mí. Bueno, ahora que lo pienso, si es rápido, me espero, que eres el único que me devuelve las pelis.

V. Blank Gogh:

Mientras la busco, ¿por qué no le cuentas algo de lo tuyo? Todo el día con tanto vídeo, tanta peli... ¡Se te va a quedar cara de pantalla! Anda...

Paco, el de la VRAM:

¡Que no solo tengo pelis, también tengo RAM! Todo el día igual... Como sea... Hola, soy Paco, director de cine. A ver, ¿qué quieres saber?

VRAM



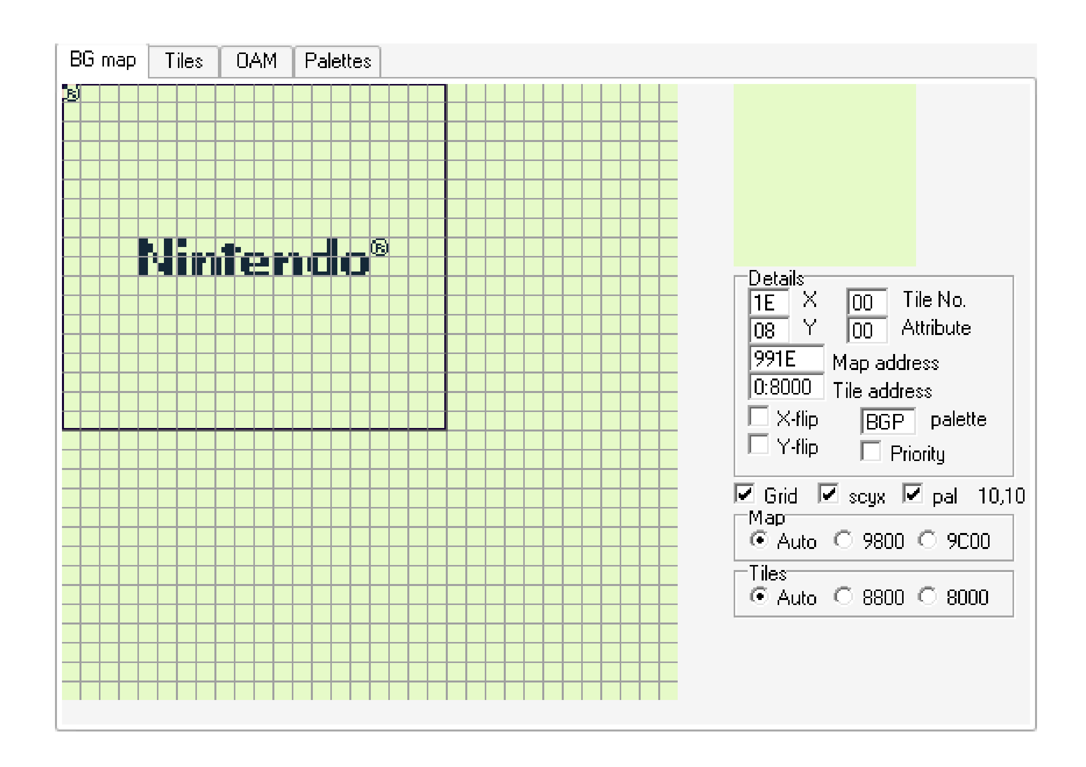
Paco, el de la VRAM:

¿QUÉ? ¿Cómo que no sabes NADA de la VRAM? Es como si me dijeras que no has visto ninguna peli de *Star Wars*... Mira, ni lo quiero ni saber. Empecemos.

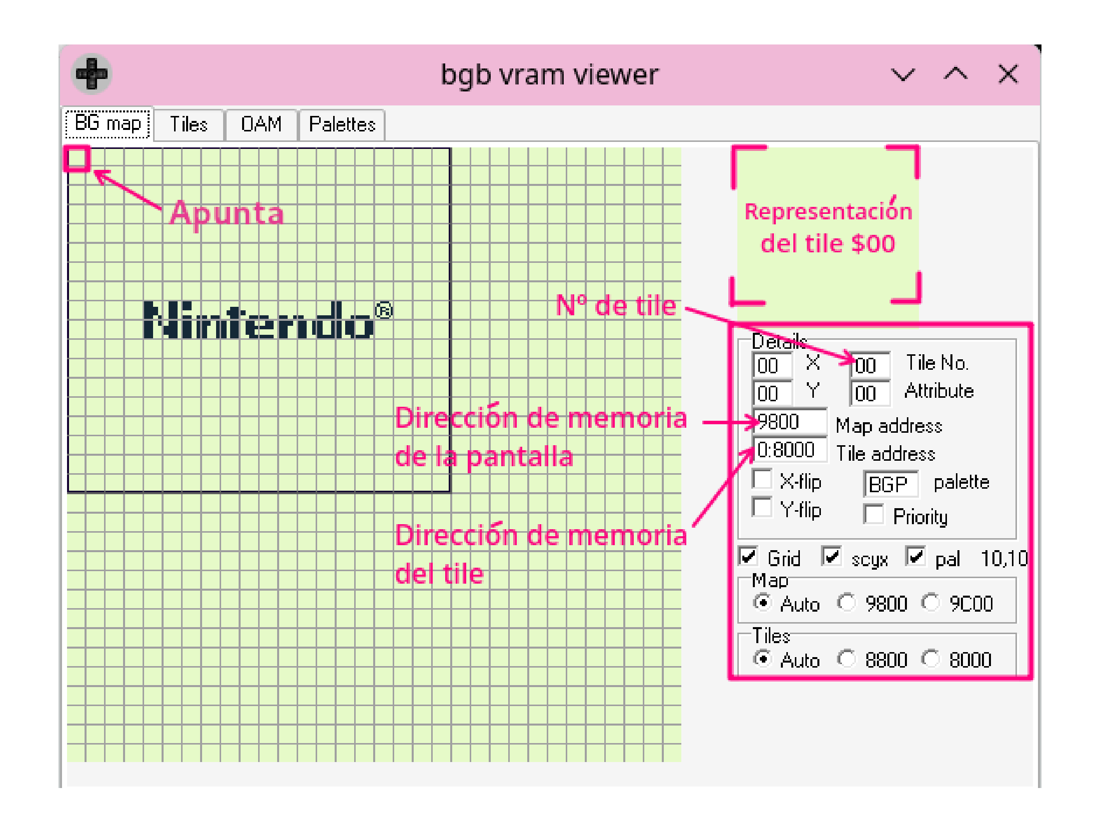
La VRAM es una región de memoria "especialita". La PPU la interpreta de manera diferente a las demás. Va de la dirección `$8000` a la `$9FFF` de la memoria. Para empezar, te interesará saber que la pantalla empieza en la dirección `$9800`. Creo que este número te debería sonar si has superado la prueba de dibujar un tile.

Todo esto que te estoy contando de boquilla está muy bien, pero se te va a quedar más claro si te lo enseño en persona. Menos mal que siempre llevo conmigo el depurador BGB.

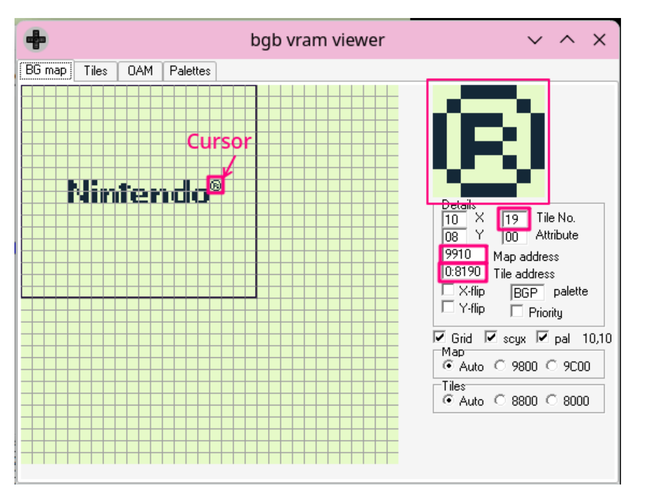
Tal vez recuerdes esta imagen de aquí:

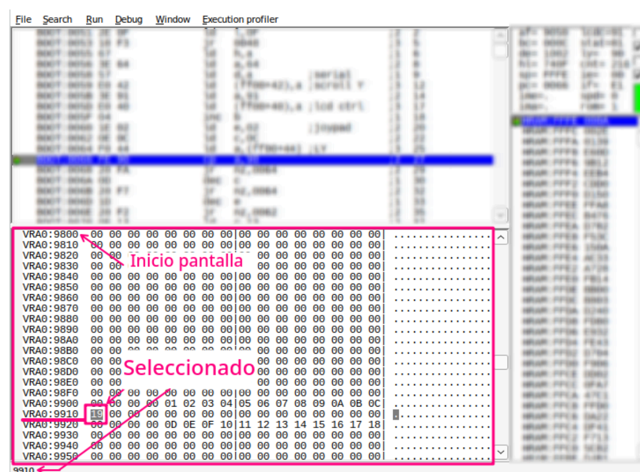


Te la explico un poco:



Conforme pases el ratón por las diferentes casillas de la rejilla, el número de la dirección de pantalla cambiará y, según qué *tile* contenga, todo lo que tenga que ver con ellos también. Te pongo un ejemplo donde señalo la ®:





Cómo ves, el *tile* es el número 19, justo el que usaste en la Prueba 1. Con el visor (izquierda) es muy fácil saber dónde tienes que colocar los *tiles*. En la imagen de la derecha, la posición que nos señala el visor tiene como contenido el número del *tile* ®.



V. Blank Gogh:

Toma, Paquito, ya he encontrado la peli.



Paco, el de la VRAM:

¡Jope! Siempre me dejáis a medias. Si no había prisa, hombre, solo me ha dado tiempo a contarle lo más básico. Podrías haber tardado un poquito más, *así te pongo una multa*...



V. Blank Gogh:

¿Qué farfullas por lo *bajini*?



Paco, el de la VRAM:

Nada, nada... Bueno, lo dicho. Me voy para el videoclub de nuevo. Artistilla, si quieres que te cuente más cosas sobre Star Wars o la VRAM, me puedes visitar en mi videoclub. Lo encontrarás en el **Edificio PPU Incertidumbre 21**. Esta semana tenemos 2 por 1 en pelis de ficción. ¡Nos vemos!

V. Blank Gogh:

Bueno, aprendiz de bohemio, ahora que ya te hemos contado la chapa y hemos saciado nuestro apetito, te voy a pedir ayuda. Tanto tiempo encerrado en el internado me ha hecho perder mis facultades artísticas. Me gustaría practicar mi trazo. A veces por la noche veo el cielo lleno de estrellas y me gustaría plasmarlo en un lienzo...

Ya que tú no sabes de arte, pero sí de Game Boy, y a mí me pasa al contrario, me preguntaba si me podrías ayudar a hacerlas. Siempre he hecho trazos cortos, a veces puntillismo..., pero quiero hacer líneas pequeñas y seguidas.

Dibuja una línea corta con *tiles*



V. Blank Gogh:

La **línea** que quiero hacer es cortita, hecha **de 5 *tiles* seguidos**. La puedes poner donde quieras de la pantalla y con el *tile* que quieras.

Yo te recomendaría empezar en una zona alejada del logo de *Nintendo* para no confundirte, por ejemplo, donde empieza la pantalla.

Como sé que superaste las pruebas, sé que lo podrás conseguir. Si tienes dudas, repasa las instrucciones que viste, trastea con el **visor de VRAM** con la pestaña de *BG Map* o *Tiles*.

Te dejo por aquí algunas ideas de lo que puedes hacer:



¿Lo has conseguido? ¡Enhorabuena! Podríamos decir que estos han sido tus primeros pasos como expresionista. Eso se merece celebrarlo, ¡ronda de bocatas para todos! ¿Cómo que estás lleno? Bueno, para bajar la comida te recomiendo dar una vuelta por el resto de Pueblo Palette. ¡Nos vemos!

Dibuja una línea larga con *tiles*



V. Blank Gogh:

Pero bueno, ¿a quién tenemos aquí? ¡Si has vuelto! ¿Te quedaste con hambre? Me pillas con mi buen amigo Henri.



Henri ???

¿Este te ha dado mucho la tabarra antes? Me lo imaginaba... Ah, con que le has ayudado con el trazo corto, ¿no? Chorradas, ahora todo el mundo sabe que se lleva pintar grandes superficies. Te voy a poner un reto interesante de verdad. Por cierto, llámame **Blanktisse**.



H. Blanktisse

Bien, como decía, me gustaría que probases a pintar una línea larga, ahí veremos de lo que es capaz un gran artista, a ver si la puedes hacer con pulso firme, como si no quisieras levantar el lápiz del papel...

Dibújame por la pantalla de la Game Boy una línea de 9 o más *tiles* seguidos. Yo probaría primero con **9** y después con muchos, como unos **19**.
Puedes pintarla en la posición que quieras de la pantalla. Si no te quieres calentar la cabeza, puedes empezar en la esquina superior izquierda.

Con 9 *tiles* seguidos **Con 19 *tiles* seguidos**





H. Blanktisse

Con 9 te falta 1 por mostrar, y con 19 te sale un hueco... No te ha salido... Vaya... ¿Por qué pones esa cara tan larga?



V. Blank Gogh:

¡Ahora no finjas compasión! Sí sabías lo que iba a pasar. Siempre igual...



H. Blanktisse

¡Ja, ja, ja! Me tienes calado. Artistilla, mis disculpas, se lo hago a todo el mundo, no es nada personal. Mira, como me has caído bien, te vamos a explicar lo que está pasando aquí.



V. Blank Gogh:

¿"*Vamos*"...?

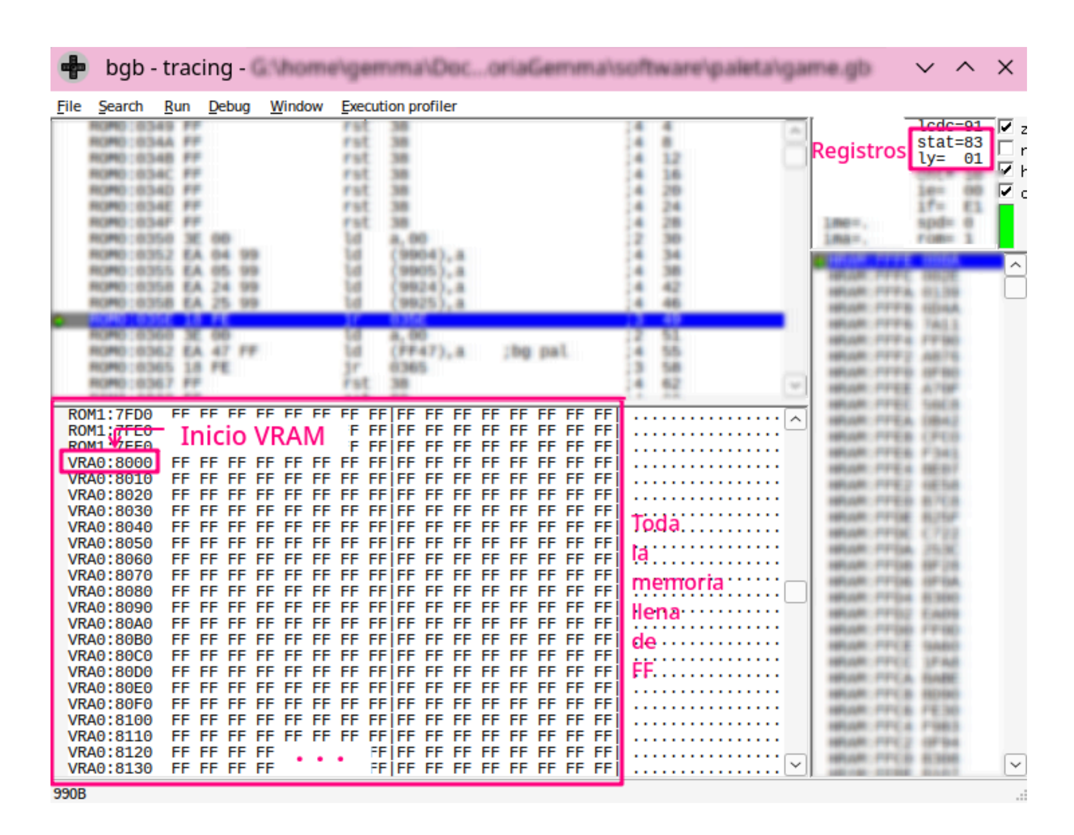
VBlank y HBlank



V. Blank Gogh:

Como sea, te cuento: acabas de ser víctima de la **VRAM inaccesible**. Para saber en detalle de qué te estamos hablando, te recomiendo que hagas una visitilla al bueno de Juan PPUesta y que te explique sobre los modos de la PPU.

Básicamente, te acabas de topar con los modos 2 y 3 durante la ejecución del programa, donde la VRAM está inaccesible. Si abres el depurador y vas pasando instrucciones (te recuerdo que es con la tecla **F7**), verás que en la zona de la memoria, p. ej., si te colocas a la altura de la VRAM, verás que en algún momento se ve como en todas las direcciones pone **FF**.





H. Blanktisse

Cuando veas esto, te tienes que fijar en los registros STAT y LY.

- STAT: te dirá en qué modo está la PPU.
- LY: te dirá por qué fila del *scanline* va el programa.

Si pasamos el `stat = 83` a binario, nos da `%10000011`. Si recordamos las palabras de Juan, estos 2 últimos bits nos dicen que la PPU está en Modo 3, por tanto, la VRAM está inaccesible, no se puede pintar nada en pantalla.



V. Blank Gogh:

Esto de abrir el depurador BGB e ir pasando de instrucción a instrucción es hacer una traza, ¡las famosas trazas!

- Nada más abrir BGB (con la tecla `ESC`): `STAT = 81` (VRAM accesible, *la verás casi toda llena de* `$00`).

- Si mantenemos pulsado `F7`, veremos cómo rápidamente va cambiando, se está cambiando de modo entre el 2 → 3 → 0 (HBlank) por cada línea (nos podemos fijar en LY).
- Cuando lleguemos a `LY = 90` (línea 144), volveremos a ver la VRAM accesible (VBlank). *Sí que has tenido paciencia para llegar a esta*...

Durante estos dos **periodos de tiempo, HBlank y VBlank**, podremos dibujar lo que queramos por pantalla.

H. Blanktisse

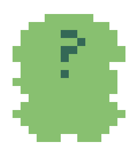
¿Que cómo solucionas ahora lo de dibujar el trazo largo? Pfff, pues... No sé. Nosotros solo sabemos señalar lo evidente, pero resolverlo... Creo que te falta algo para poder lograrlo. En el pueblo de al lado se comenta que saben cosas bastante útiles para estos casos.

V. Blank Gogh:

Si pudieras tener alguna manera de detectar qué marca STAT y poder comprobarlo muchas veces seguidas... Habrá que darle alguna vuelta, *o muchas*, vueltas. Artistilla, no insistas, que nosotros no sabemos resolverlo. Ya vendrás en algún momento y nos lo explicarás. Te esperaré con un bocata y los brazos abiertos. ¡Chao!

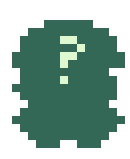
Jardines del Museo

¡Qué buen día hace! Qué gusto pasear por estos maravillosos jardines, están preciosos. Muchos artistas están plasmando su belleza en sus lienzos. Es un remanso de tranquilidad, de no ser por unos cuchicheos y risitas entre los arbustos. Te acercas a ver de qué se trata.



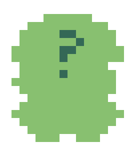
Cuchicheo 1:

Que te he dicho que lo he visto pasear por aquí; tenemos que avisar al resto ya.



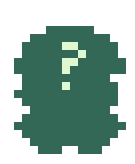
Cuchicheo 2:

Y yo te he dicho que lo haría, ¡pero están muy lejos! Y hay mucho ruido... No van a oír la señal a esta distancia.



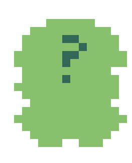
Cuchicheo 1:

Pues usa la Game Boy, que para algo la tenemos.



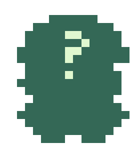
Cuchicheo 2:

¡No sé cómo usarla! Es Juanete quien sabe programar. Yo me sé el código, pero haciendo palmas. ¿Y por qué no la usas tú?



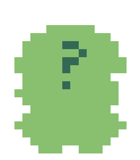
Cuchicheo 1:

Pues porque yo tampoco sé, yo solo he venido para ver qué pintaban los *Pixel Artists*, a ver si cojo inspiración.



Cuchicheo 2:

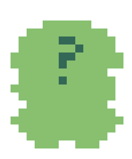
¿Y ahora qué hacemos? Si nos quedamos aquí, o les pillan a ellos o a nosotros.



Cuchicheo 1:

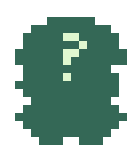
¡Yo qué sé! Ni que fuera a venir nadie que sepa de Game Boy a ayudarnos... Espera, ¡Eh, tú! ¡Quieto *parao*'!

Dos chavalitos salen de su escondite señalándote y apuntando a la Game Boy que llevas encima.



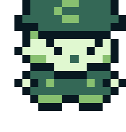
Cuchicheo 1:

Esa Game Boy... Por casualidad tú no sabrás programar, ¿verdad? ¿Sí? ¿Nos podrías ayudar? Mira, estamos en una misión ultrasecreta que tenemos que comunic…



Cuchicheo 2:

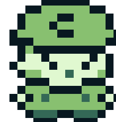
Mira que tienes morro, pero preséntate primero, melón. Perdónalo, tiene incontinencia verbal. Yo soy Corchete y este es mi hermano Caballete. Somos los hermanos Palette.



Corchete:

Cómo intentaba explicar atolondradamente mi impaciente hermano, necesitamos ayuda de alguien que sepa programar en Game Boy para mostrar un mensaje secreto por pantalla. ¿Nos echas un cable?

Código Morse



Caballete:

Bueno, como decía, tenemos que mandar un mensaje a nuestros hermanos que están al otro lado del jardín, que están en nuestra supermegaultrasecreta misión, que bajo ningún concepto te puedo decir de qué se trata.

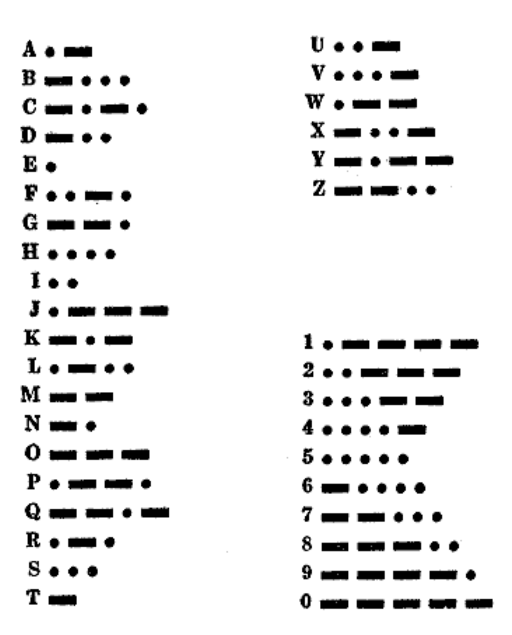


Corchete:

Le estamos gastando una broma a los abuelos que están jugando a la petanca.

¿Ves al final de este camino del jardín? Ahí están nuestros hermanos, pero con el ruido de la gente y los pájaros no escuchan nuestra señal. Nos han pedido que patrullemos la zona antes de que vuelvan los abuelos después de su paseo a jugar.

Nuestra señal es en código Morse; lo usamos para comunicar la inicial del abuelo que está al acecho. El Morse, como puedes ver, es un lenguaje binario. Ni se te ocurra decírselo al Abuelo Binario, que si no, se le va a subir el ego por las nubes de que le demos uso en algo. Te dejo una chuleta: `mira la imagen de la derecha →`

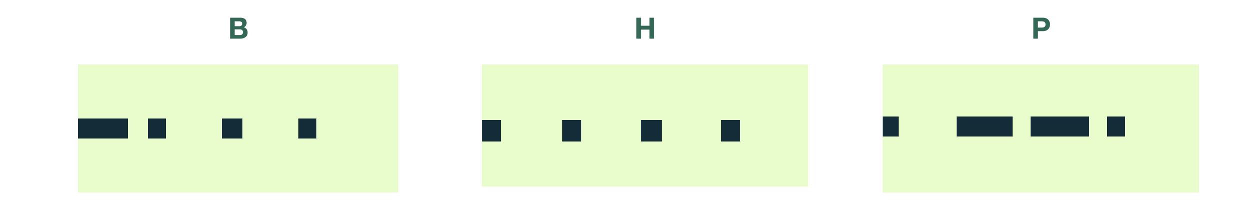


Estamos evitando al abuelo **B**inario, al abuelo **H**exa y al abuelo **P**ruebas.

Necesitamos que nos enseñes a **escribir cada letra en Morse por la pantalla de la Game Boy**. Que sea una a la vez; cada abuelo va a su bola y no creo que coincidan.

Pista: Juanete nos ha chivado que los mejores *tiles* para que se vea el ‘`.`’ y la ‘`_`’ son los `$09` y `$0C` respectivamente.

Una vez lo tengas, te debe de quedar así:





Caballete:

¿Que por qué les estamos haciendo una broma a los abuelos? ¡Porque son muy cascarrabias! (*Y nos encanta fastidiar*).

El abuelo Binario y Hexa están todo el día discutiendo sobre cuál de los dos es mejor. Si nosotros vamos a chincharles con que lo que está de moda ahora es el decimal, nos echan un sermón que, ya puedes huir, que no te los quitas de encima.



Corchete:

Y el abuelo Pruebas es un pesado; como te pille por banda, no te suelta hasta que no le resuelvas el ejercicio, que no quiero aprender a programar, ¡quiero ser músico! Que programe Juanete, que para algo sabe.

Caballete:

Ahora, con lo que nos has enseñado, será mucho más fácil patrullar. ¡Muchas gracias! Pásate (si no lo has hecho ya) cuando quieras por nuestra casa y te enseñamos qué más tácticas ultrasecretas tenemos.

Si seguimos paseando por el jardín, veremos varias personas con las que poder hablar:

- Da Bitcci te explicará la profundidad de bits.
  - **Pasa a la página 76.**
- Pixel Artist Gemma sabe cosas sobre los retos.
  - **Pasa a la página 77.**
- Los abuelos Binario, Hexa y Pruebas están dando un paseo.
  - **Pasa a la página 78.**

BPP

Da Bitci:

Este espacio es inspirador, ¿verdad? *Mi ricorda casa mia*... No sé si me apetece más ponerme a pintar, a prototipar inventos, escribir un poema, filosofar... Bien, creo que si te has parado a hablar conmigo debe ser porque quieres que te cuente algo de artista. Mira, te voy a contar algo que creo que aún no te habrán contado. ¿Conoces la **profundidad de bit**?

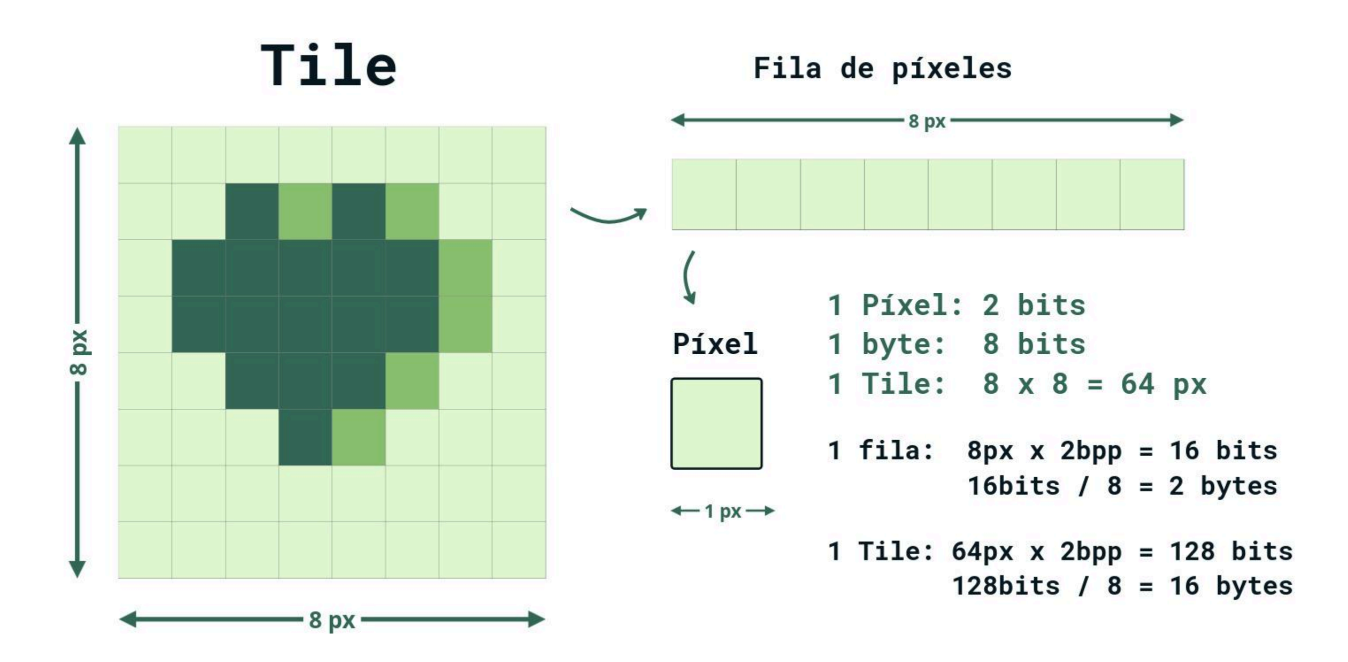
Si has visitado a los hermanos Palette en su casa, puede que te la hayan mencionado ya. La profundidad de bits, en este contexto, la usaremos para referirnos al formato gráfico con el que se codifican los colores de la Game Boy. Estos tienen una profundidad de 2 **bits por píxel**, *2 bpp*.

Todos los colores que puede representar la Game Boy se pueden codificar en 2 bits (te sonará una tablita de los índices y **colores** de la GB), por eso el 2 bpp. Ocurre para cada píxel que tiene la pantalla.

Recuerda que en GB la unidad gráfica más pequeña que se puede manejar son los *tiles*. Entonces, debemos codificar todos los píxeles de los *tiles*. Te dejo unos datos:

- Cada píxel ocupa 2 bits.
- Un *tile* tiene 8x8 píxeles = 64 píxeles.

Si las matemáticas no me fallan, 2 bits x 64 píxeles = 128 bits = 16 bytes.
Así que **1 solo *tile* ocupa 16 bytes en memoria**. Cada fila de un *tile* ocupa 2 bytes, 8 filas x 2 = 16. Por si no te ha quedado claro, a continuación tienes un esquema que te lo explica:



No te quiero abrumar con tanto tamaño ni cálculo. Tú quédate con que los *tiles* en GB tienen 2 bpp y que por eso pesan 16 bytes cada uno. En el museo te lo pueden destripar más aún, si tienes curiosidad. ¡Nos vemos!

Soluciones alternativas



***Pixel Artist* Gemma:**

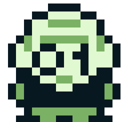
Un buen artista es aquel que, una vez ha terminado su obra, es capaz de analizarla y saber qué cosas podría mejorar e implementarlas mejor a la próxima…

¿Me dejas echar una ojeada a las actividades que has hecho en la Game Boy? Hmm... Ciertamente, has hecho un gran trabajo, pero recuerda lo que te he dicho: siempre hay una forma de hacer las cosas mejor, más eficiente... Tenlo en cuenta.

Por cierto, en el pueblo de al lado, Pueblo Bandera, tienen una feria divertida, espectacular, inspiradora... ¡Yo no me la perdería! Seguro que aprendes algo que puedes usar en lo que ya sabes hacer; a veces la inspiración te pilla en donde menos te lo esperas. A mí me pilló en la Noria de Loopita.

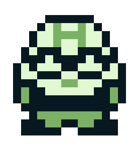
Si me disculpas, voy a seguir con este *tile* que tengo a medias. ¡Hasta la próxima!

Chascarrillos



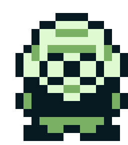
Abuelo Binario:

Pero tú te crees, esos niños del demonio, hacernos el cambiazo, ¡lo pusimos todo perdido! Y encima nos quedamos sin jugar... Como los pille... les va a caer la bronca de los `%1100100` años, ¡se van a enterar!



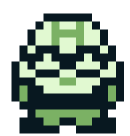
Abuelo Hexa:

Ja, ja, ja, ¡tendrías que haberte visto la cara! Mira que no diferenciar un huevo de la bola de petanca...



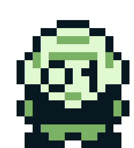
Abuelo Pruebas:

Hexa... Hexa... No chinches, que aún cobras.



Abuelo Hexa:

¡Si tú no podías parar de reír tampoco!



Abuelo Binario:

Tanto que te ríes, y los chiquillos han usado un código binario para comunicarse, así que *no te rías del mal del vecino, que el tuyo viene de camino*...

Museo de Tiles

Espacio dedicado a la belleza y al conocimiento. Al entrar por la pixelada puerta, te embriaga una sensación de paz y tranquilidad... De no ser por la figura energética que se acerca hacia ti a toda prisa.

???:

¡Saludos, joven artista! Es para mí un gran honor darte la bienvenida al Museo de *Tiles* de Pueblo Palette. Me presento, mi nombre es Tilina Pombalina, la directora del Museo. Blank Gogh ya me había avisado que vendrías, ¡qué ilusión un nuevo visitante! La gente del pueblo ya ha venido tantas veces que nunca puedo hacer ninguna visita guiada... ¡Menos mal que has venido! Empecemos...

Tiles



Directora Pombalina:

¿Qué es un *tile*? A lo mejor te has paseado por toda la ciudad y nadie te lo ha explicado.

Un *tile* es la unidad gráfica más pequeña que puede manejar la Game Boy. Está formado por 8x8 píxeles cada uno, dispuestos en forma cuadrada (8 filas y 8 columnas). Son como una especie de *azulejos* que pones para decorar una superficie, en nuestro caso, la pantalla de la GB.

Los *tiles* se pueden mostrar como parte del fondo (*background*), como *sprites* (objetos) o como ventana (*window*).

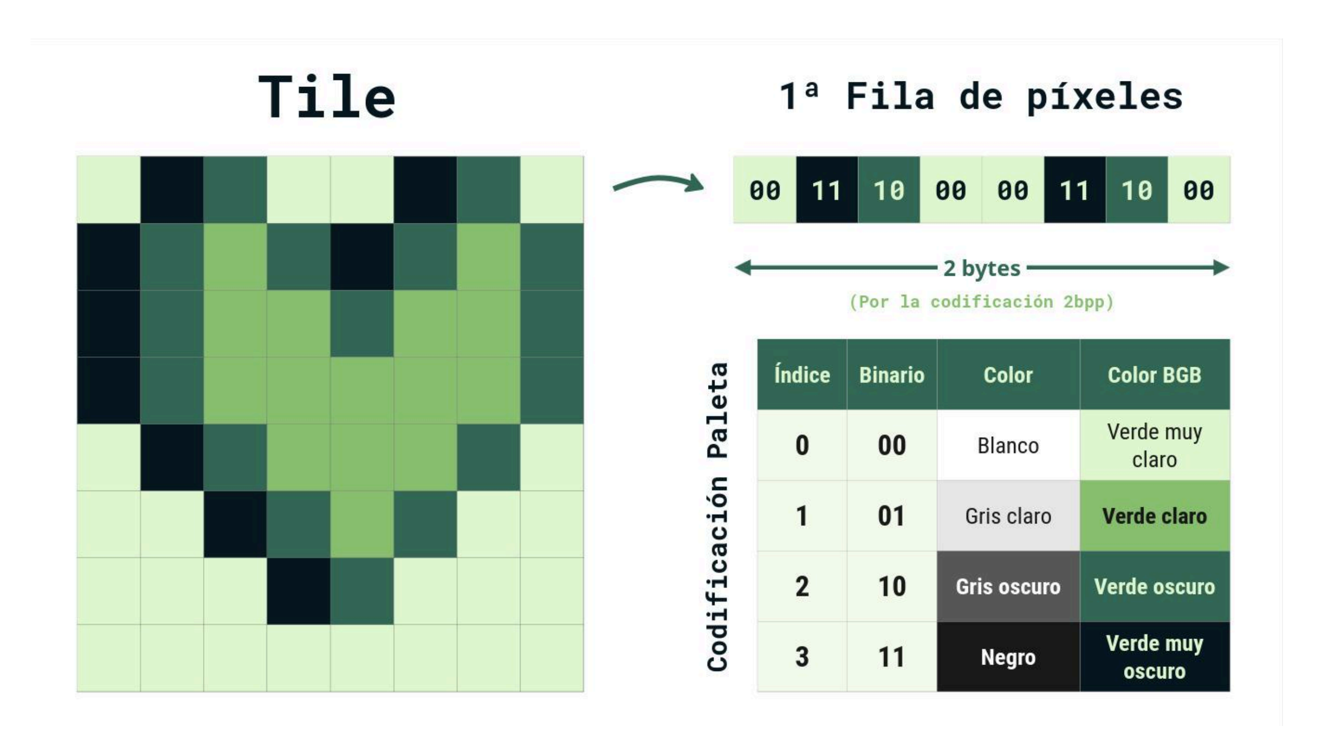
Toda la información de los *tiles* se almacena en la VRAM. Si te has encontrado a Paco puede que te suene. Esta información la puedes encontrar de `$8000` a la `$97FF`.

Codificación de los *tiles*



Directora Pombalina:

Puede que te suene que los *tiles* tienen 2 bpp y que cada uno ocupa 16 bytes. Cada fila de un *tile* se codifica con 2 bytes. Fíjate en este ejemplo de *tile* de corazón:



Índice	Binario	Color	Color BGB
0	00	Blanco	Verde muy claro
1	01	Gris claro	Verde claro
2	10	Gris oscuro	Verde oscuro
3	11	Negro	Verde muy oscuro

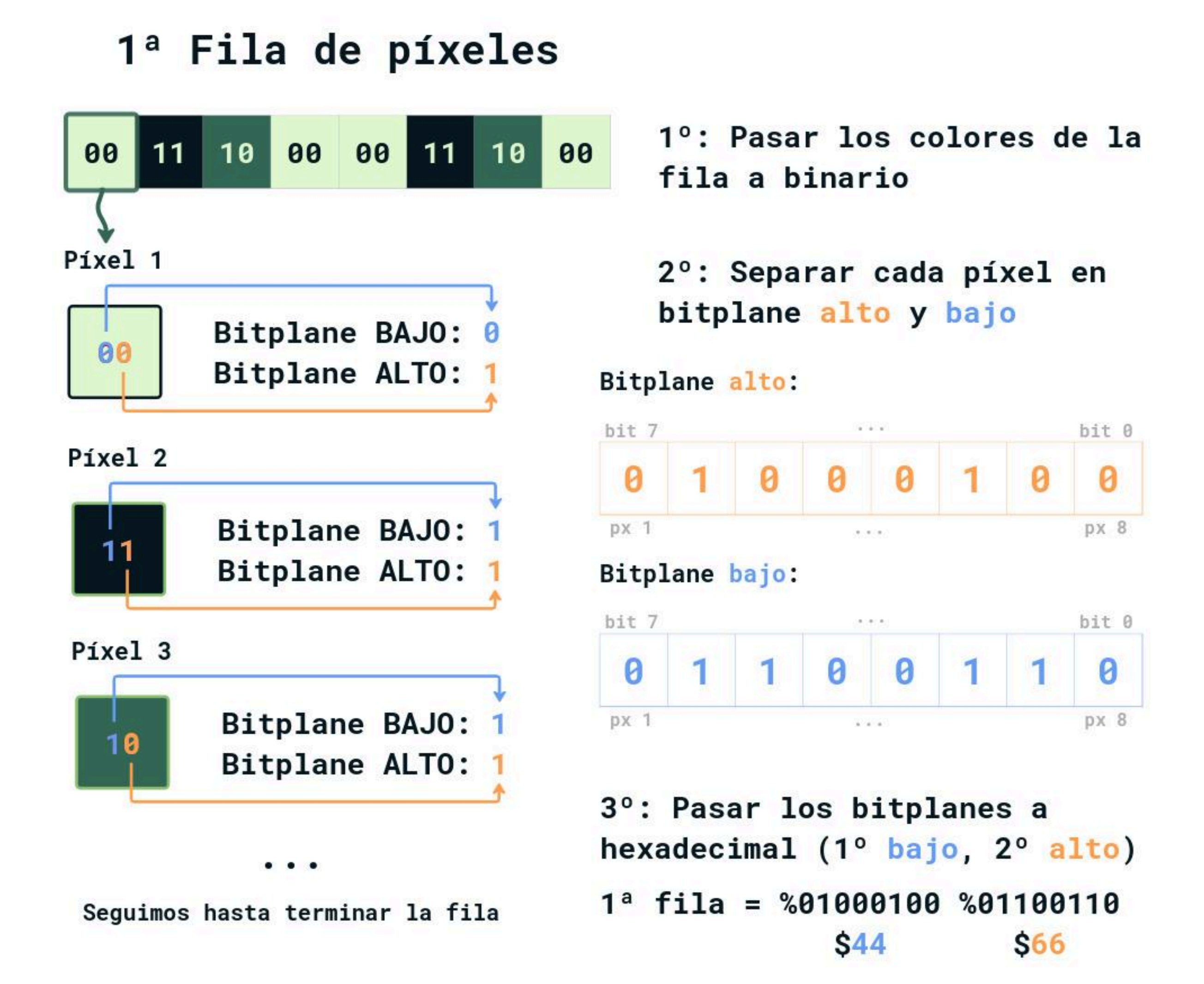
Podrías pensar que el primer byte codificaría los 4 primeros píxeles, y el segundo, los otros 4. Pero nada más lejos de la realidad:

- El 1.º especifica el bit menos significativo del color de cada píxel.
- El 2.º especifica el más significativo.

Entonces, tenemos que con partes de cada pareja de bytes tenemos la codificación de los píxeles. A esto se le llama ***bitplane***. Tendremos el *bitplane* alto y el *bitplane* bajo para cada fila:

- El *bitplane* **BAJO** es el primer byte: especifica los bits **menos** significativos.
- El *bitplane* **ALTO** es el segundo byte: especifica los **más** significativos.

Siguiendo con el ejemplo anterior, separamos los bits de cada píxel en su correspondiente *bitplane*:



Para saber cuál es el primer píxel de una fila, hacemos:

- Pasamos cada píxel a su correspondiente color en binario.
- Separamos los bits del píxel en los *bitplane* alto y bajo. Lo repetimos para cada píxel de la fila.
    - El bit de la **izquierda** corresponde al bitplane **bajo**
    - El bit de la **derecha** corresponde al **alto**.
- Pasamos los *bitplanes* resultantes a hexadecimal. El primer byte lo conforma el *bitplane* bajo y el segundo el *bitplane* alto.

Paseando por las salas del museo, llegas a una donde hay una mujer de cabello negro como el azabache y el pelo adornado de flores mirando unos cuadros. Te acercas a hablar con ella.

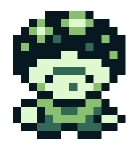


FFrida Kahl00:

Buenas, joven artista, ¿qué te trae por aquí? ¿Estos cuadros maravillosos? Y que lo digas... Mira este de aquí, ¿ves algo raro? ¿No? Te voy a contar un secreto, aquí antes había un manchurrón... Te lo aseguro por experiencia, lo pinté yo.

Así que quieres que te cuente cómo lo limpié... Permíteme corregirte: más que limpiar o borrar, el mejor término para describirlo es que lo tapé, lo sobreescribí. ¿Sabes cómo tapamos aquí las cosas? ¡Con *Tile*pex!

*Tile*pex



FFrida Kahl00:

Puede que haya exagerado un poco antes, nadie le llama *Tile*pex salvo yo.

El caso: cuando tienes algo que has dibujado por la pantalla de la Game Boy y quieres "limpiar" el dibujo, es tan fácil como **sobreescribir** la dirección de memoria donde se encuentra ese *tile* por el *tile* que quieras.

Normalmente, usarás el `$00`, que es el que no tiene nada, un color de fondo plano, como si fuera un **típex**. Sencillo, ¿verdad?

Para demostrarme que lo has pillado, te propongo que hagas lo siguiente: escríbeme un código que pueda **mostrar el logo de Nintendo sin la N**. ¡Aplícale *tile*pex!

Te debe quedar algo así:





FFrida Kahl00:

¿Lo conseguiste? ¡Qué bien! Recuerda: ahora podrás usar el *Tile*pex siempre que quieras quitar algo de la pantalla.

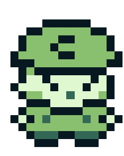
Casa de los hermanos Palette

Hogar de los traviesos hermanos Palette. Desde fuera se puede entrever, como si la casita de los osos de Ricitos de Oro se tratase, que todo está por cuatro y colorido. Hay cuatro camitas, cuatro sillitas, cuatro tazones de sopa... que van de color verde clarito a oscuro.



Corchete:

... Y entonces conseguimos usar la señal por los pelos. Menos mal que Colorete la vio, ¿verdad, Caballete?



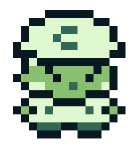
Caballete:

Justo el abuelo Hexa había pasado por detrás de nosotros y no lo teníamos localizado hasta que casi estaba encima de vosotros.



Corchete:

Me entraron los mil males por si nos pillaba. Qué suerte que Juanete hizo el cambiazo de las bolas de petanca por los huevos antes de que nadie se percatara.



Juanete:

¿Visteis la cara que se les quedó? Lo más difícil de todo fue reírme en voz alta mientras se enfadaban. Tuvimos que salir por patas. ¡Buen trabajo, equipo!

* Toc, toc *



Juanete:

¡Alto! Que nadie se levante a abrir la puerta. Revisad que no sean los abuelos enfadados.

Esperas detrás de la puerta donde destacan 4 mirillas con ojos curiosos al otro lado.



Caballete:

¡Pero si es nuestro compinche! Te damos la bienvenida a nuestro hogar. Juan, como te espabiles, te va a quitar el trabajo: sabe programar.



Juanete:

¿Sí? Venga ya, apuesto a que no sabe nada sobre la **paleta**. Vaya, parece que he acertado, ja, ja, ja. Era broma. Si quieres, te cuento un poco de lo que sé. Colorete, ven a echarme una mano *(quedaría fatal si me equivoco)*.

La paleta



Juanete:

Una paleta es un *array* de colores. La Game Boy usa una paleta de 4 colores.

Los *tiles* cogen los colores de la paleta, de una manera parecida a un libro para colorear: el *tile* tiene unos índices y estos corresponden a un color u otro de la paleta.

La codificación del **color** va del 0 al 3 en binario:

Índice	Binario	Color	Color BGB
0	00	Blanco	**Verde muy claro**
1	01	Gris claro	**Verde claro**
2	10	Gris oscuro	**Verde oscuro**
3	11	Negro	**Verde muy oscuro**

Se llama **codificación 2 bpp**. Puede que ya te la haya explicado algún artista del jardín, pero si aún no te has encontrado con ninguno, te recomiendo que los busques por el Jardín o por el Museo. De momento, con que te quedes con la tablita, va perfecto.

La Game Boy tiene 3 paletas: una para el fondo, `BGP`, *BackGround Palette*, y otras 2 para los objetos, `OBP0` y `OBP1`, *OBject Palette*. De momento con la primera *vamos que chutamos*.

Acceder a la paleta es muy sencillo: como solo vamos a manipular la **BGP**, solo nos tenemos que acordar de 1 dirección de memoria, la **$FF47**. Al ser una dirección de memoria, caben 8 bits, 1 byte.

De los 8 bits, **cada par de bits codifica un color**. Para determinar el color usamos los que tenemos establecidos antes:

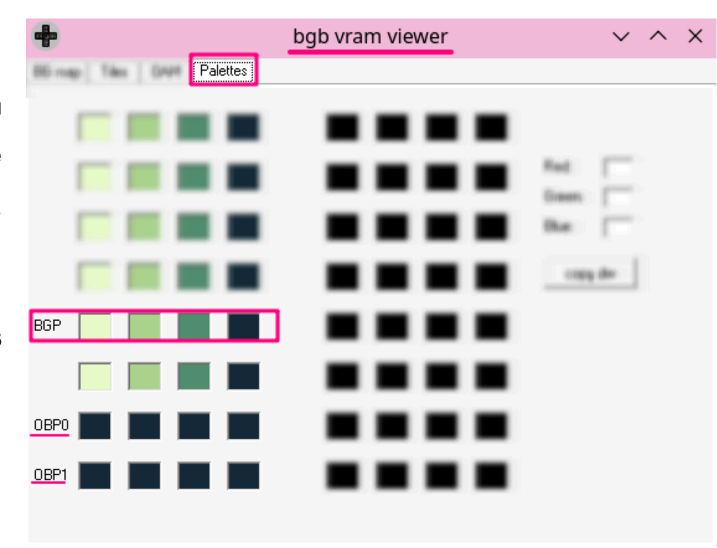
Bit	7 6	5 4	3 2	1 0
Color (id)	3	2	1	0
Binario	11	10	01	00
Hexadecimal	E		4	

En este ejemplo, el contenido de `$FF47` sería `$E4` y saldrían los 4 colores disponibles.

Colorete:

Juan, todo muy bien explicado, pero si no le enseñas algo a la criatura, no se va a acordar.

Permíteme que te diga dónde lo puedes encontrar en el depurador BGB.

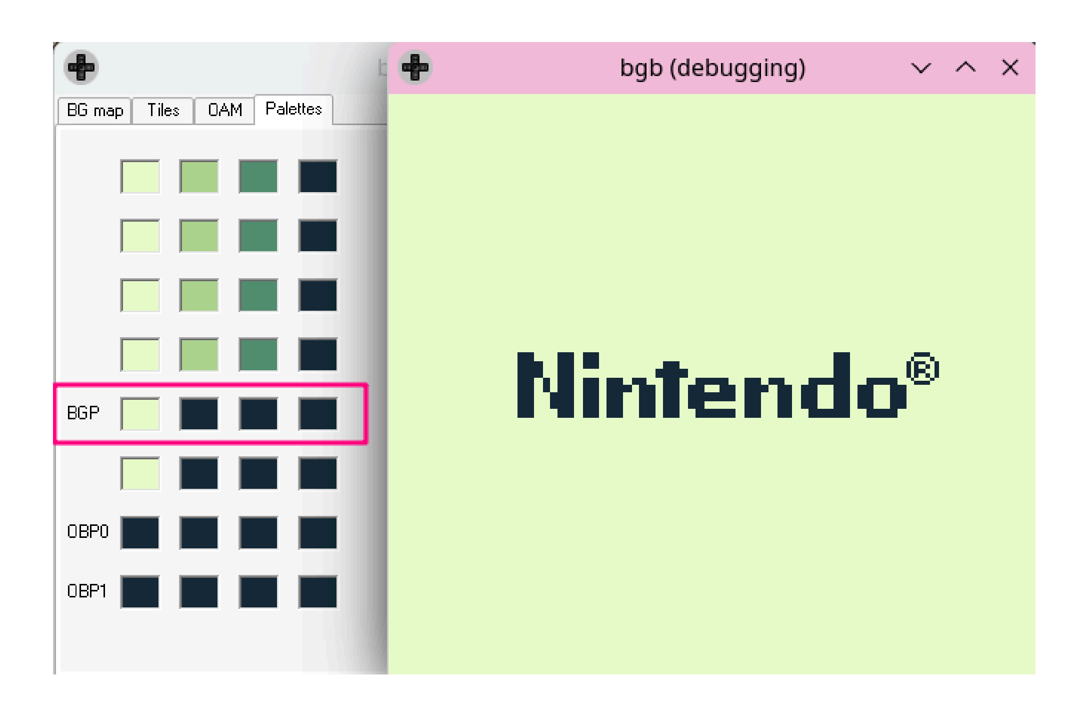


Una vez más, para acceder a este menú, es entrando en el *VRAM Viewer*. Puedes hacerlo con:

- [ESC + F5].
- [ESC y navegar en *Window* < *VRAM Viewer*].

Una vez dentro, es entrar en la pestaña de **Palettes** y puedes ver cómo tienes configuradas las paletas.

En este ejemplo salen 4 colores disponibles en la BGP, pero lo más normal que te encuentres si no has modificado nada en `$FF47` es que te salga así:



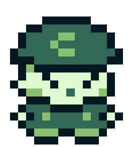
Juanete:

El cambio o modificación de la paleta permite hacer algunos efectos en lo que estamos viendo por pantalla, p. ej., efectos de parpadeo cuando un personaje sufre daño, *fades in* o *out* en la pantalla... Está bien que sepas que podrás hacer muchas cosas con ellas, pero no adelantemos acontecimientos. Esto lo veremos más adelante.

Semáforo Palette

Colorete:

La teoría está muy bien, pero nosotros aprendemos para darle un uso práctico. Te presentamos nuestra nueva idea de técnica ultrasecreta de comunicación para cuando estemos en alguna misión: el **semáforo palette**.



Corchete:

Chicos, gastamos bromas, no somos agentes secretos...

Caballete:

Este nuevo ingenio consiste en que, para tener más señales para comunicarnos, *a veces con el Morse se complica la cosa*, hemos pensado en enviarnos señales de colores como si la Game Boy fuera un semáforo.

Para ello queremos **modificar la paleta de manera que toda la pantalla se tiña de un color**. Como solo tenemos una Game Boy y ahora la tenemos cargando, ¿te importa programarlo en la tuya?

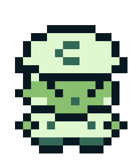


Corchete:

¿Este tiene siempre tanta cara?



Juanete:

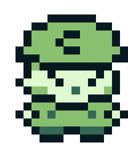


El sistema lo podremos combinar con las letras Morse. Podremos indicar el nivel de peligro o proximidad de los abuelos con el semáforo y con el Morse especificar cuál viene. El semáforo será el siguiente:

- Color **0**, *verde muy clarito*: no hay peligro, tranquilidad.
- Color **1**, *verde claro*: precaución moderada, están lejos.
- Color **2**, *verde oscuro*: precaución considerable, están cerquita o se están acercando.
- Color **3**, *verde muy oscuro*: peligro inminente, salir por patas.

¿Nos puedes sacar qué paletas nos hace falta cargar para cada color? Y ya que estás, muéstranoslas por pantalla. Te tiene que salir algo así:

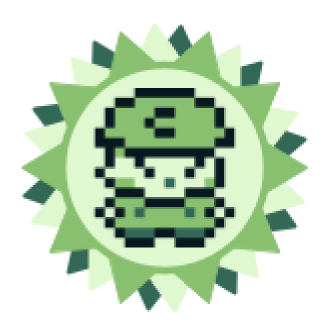




Caballete:

¡Has vuelto a demostrar que eres una máquina! ¡Muchas gracias por tu ayuda! La próxima trastada te la dedicaremos :)

Tras completar las técnicas de Código Morse y Semáforo Palette obtienes el reconocimiento de Bormista de 1ª.



¡Has conseguido el Logro Bromista de 1ª!

Parte 3

En esta parte continuaremos abordando más conceptos esenciales para el desarrollo eficiente de programas en ensamblador para Game Boy. Dominar estos mecanismos es crucial para escribir código robusto, claro y optimizado. Trataremos los siguientes temas:

- **Flags:** Aprenderemos cómo las instrucciones afectan los indicadores de estado (*flags*) del registro F del procesador (z, n, h y c) y cómo utilizarlos para controlar el flujo de ejecución.
- **Bucles:** Veremos cómo implementar estructuras de repetición controladas mediante saltos condicionales (`jr`, `jp`), esenciales para tareas iterativas.
- **Etiquetas:** Estudiaremos cómo emplear identificadores simbólicos (etiquetas) como puntos de referencia en el código, facilitando saltos (`jr`, `jp`, `call`) de manera segura y sostenible.
- **Funciones:** Aprenderemos a crear bloques de código reutilizables (funciones), aplicando las instrucciones `call` y `ret`. También veremos el manejo y comportamiento de la pila al usar las instrucciones `push` y `pop` para preservar el estado del programa.
- **Máscaras de bits:** Introduciremos el uso de operaciones lógicas (`AND`, `OR`, `XOR`) para manipular bits específicos dentro de registros mediante máscaras de bits, una técnica fundamental para el control del hardware de nuestra consola.

Estos conceptos no sólo son fundamentales para comprender la arquitectura interna del sistema, sino que también sientan las bases para escribir programas más modulares, seguros y eficientes.

Al finalizar este capítulo, serás capaz de diseñar estructuras de control complejas, optimizar operaciones de bajo nivel, y manipular directamente la memoria y el hardware de la Game Boy con precisión.

índice del capítulo

Todos los sprites e ilustraciones de la Parte 3 - Pueblo Bandera han sido diseñados y creados por Manuel Alejando López Pomares



Pueblo Bandera



????:

¡Bienvenido al **Pueblo Bandera**, bitsitante! ¿Has venido a ver la feria? ¡Por supuesto, qué vendrías a hacer si no a éste pueblo tan alejado de la civilización! Este año el ambiente está muy animado.

¿Eh? ¿Que qué es esa carpa tan grande de allí? ¡Un maravilloso circo, bitsitante!

Pero cuidado, **podrán entrar al circo tan solo aquellos que tengan todas las fichas de esta feria**.

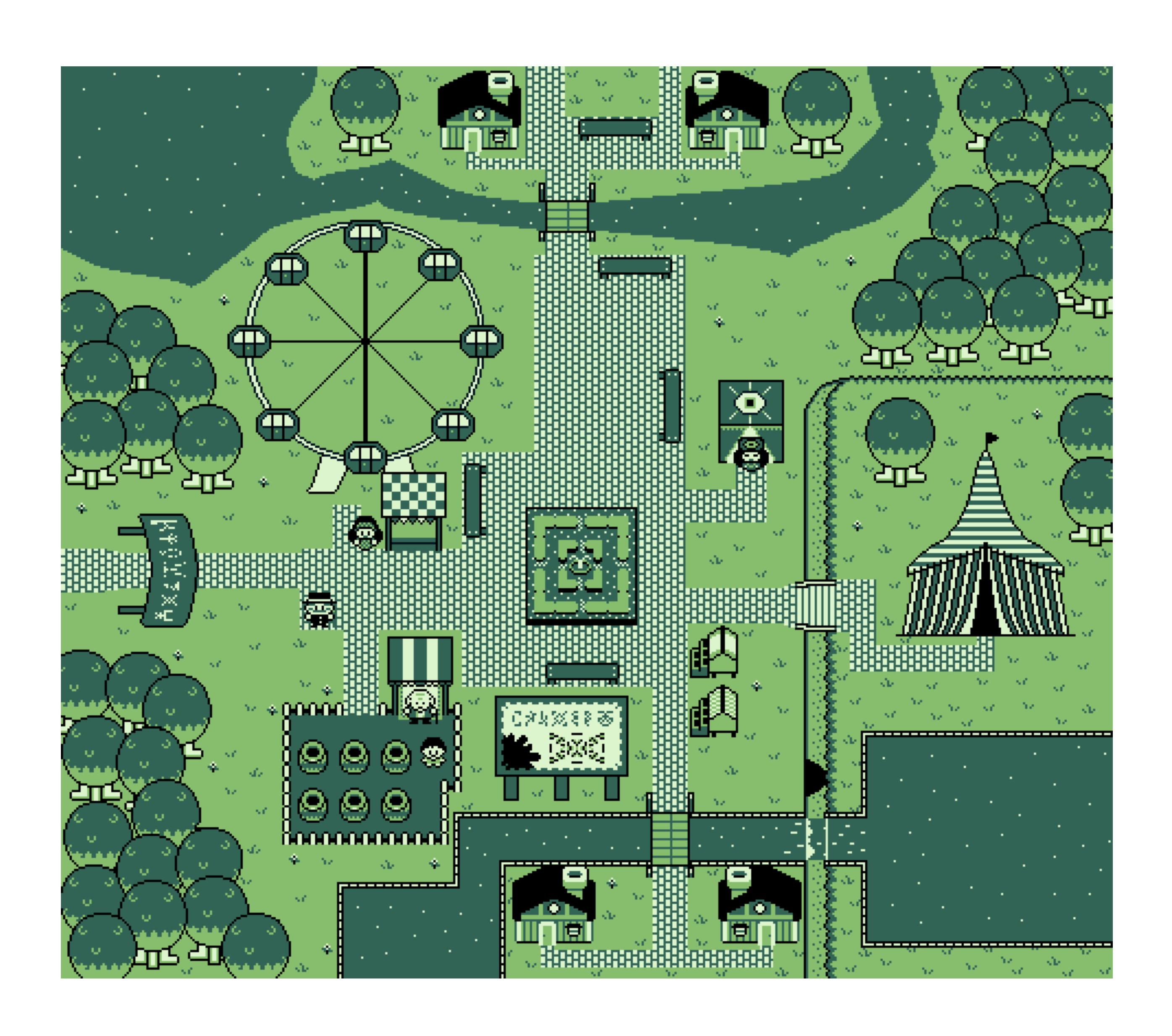
¿Qué son **las fichas**? Son unos pequeños trofeos por participar en los distintos garitos.

Mi garito favorito es **El garito de Flaguelito** ¡Qué nostalgia! Lo solía frecuentar mucho en mi juventud. Te recomiendo empezar la visita por ahí.

¡Pasa un bonito día!

Has llegado a Pueblo Bandera, parece que justo a tiempo para disfrutar de su famosa feria. El pueblo está muy animado. ¿Qué quieres hacer?

- ➢ **[Para ir al garito de Flaguelito, pasa a la página 99]**
- ➢ **[Para ir a la noria de Loopita, pasa a la página 109]**
- ➢ **[Para ver a la adivina Madame Labelette, pasa a la página 117]**
- ➢ **[Para ir al circo de Funcelmo, pasa a la página 123]**
- ➢ **[Para entrar en la grieta paleontológica, pasa a la página 133]**
- ➢ **[Para ver el panel de operaciones, pasa a la página 143]**



El garito de Flaguelito



Flaguelito:

¡Ah, un bitsitante! Te doy la bienvenida, pasa pasa.

Soy Flaguelito, ¿es la primera vez que vienes a mi garito verdad? Otra cosa puede que no pero tengo muy buena memoria, je je.

La pasión de mi abuelo Flagbio era romper jarrones. Los fabricaba con sus propias manos para después hacerlos añicos como si de piñatas se tratara.

Disfrutaba tanto que quería compartir su dicha con los demás, así que construyó éste garito para que todo el mundo pudiera romper jarrones en armonía.

¿Te apetece probar? Te aseguro que una vez lo pruebas ya no querrás hacer otra cosa.

Por supuesto, como es tu primera vez será gratis. ¡Ven por aquí que te explico las bases!

El Registro F



Flaguelito:

¿Me has dicho que vienes de **Pueblo Game Boy** verdad? Entonces ya sabrás lo que son **los registros**.

Te voy a presentar un registro nuevo, **el registro F** o también llamado **registro de Flags** (Banderas). **Es un registro de 8 bits**, concretamente los 8 bits más bajos del **registro AF,** de 16 bits.

Te preguntarás ¿y qué son los flags? Muy sencillo, son como unos pequeños interruptores que **sólo pueden almacenar el valor de 1 bit**, es decir que solo pueden tener de valor un 1 (encendido) o un 0 (apagado).

Existen distintos flags y cada uno espera a que se cumpla cierta condición. **Si la condición se cumple** su valor será positivo, es decir será un 1, y **si no se cumple** será negativo, es decir será un 0.

A mi me gusta compararlo con aquel juego de golpear topos. Miras al agujero ¿hay algún topo? Si no hay nada (0) pasas al siguiente agujero.

Sin embargo, si descubrimos que hay un topo asomando (1) tomamos esta información para desencadenar algún tipo de acción... que en este caso suele ser golpear en la cabeza al pobre bicho.

Los distintos flags de los que te hablo son los son los flags **z**, **n**, **h** y **c**. Todos ellos están representados en los 4 bits más significativos del registro F.

Déjame que te haga un pequeño croquis, quizás así se entienda mejor. La estructura del registro F es algo tal que así:

Registro F								
NºBit	7	6	5	4	3	2	1	0
Flag	z	n	h	c				



Consejo del profesor:

"Recuerda que las posiciones de los bits se cuentan de izquierda a derecha, empezando por 0."

Cada uno de los flags tiene las siguientes condiciones para activarse:

Flag	Significado	Descripción
z	*Zero*	Se activa si el resultado de la última operación es cero.
n	*Substraction*	Se activa si la última operación ha sido una resta.
h	*Half-Carry*	Se activa si en la última operación ha habido acarreo desde los cuatro bits bajos a los altos.
c	*Carry*	Se activa si ha habido overflow en la última operación (el resultado tras una suma es mayor que `$FF`, o tras una resta es menor que `$00`).

¿Sencillo no? Muy bien, continuemos con el registro F.

A diferencia de otros, **el registro F es de solo lectura**. Ésto quiere decir que no puedes modificar su contenido manualmente intentando cargar valores en él. Sin embargo, existen algunas instrucciones en **RGBASM** que alteran su valor.

¿Qué dices? ¿Que cómo que algunas? Pues si, **no todas las instrucciones alteran los valores de los flags**. Así que debes prestar especial atención a las instrucciones que estás usando.

Algunos ejemplos de instrucciones que **si alteran los flags** son:

- Las operaciones aritméticas `ADD`, `SUB`, `INC` y `DEC`.
- Las operaciones lógicas `AND`, `OR`, `XOR`.
- La operación de comparación `CP`, que no modifica el registro A, pero sí los flags.

Veamos un ejemplo usando algunas de estas instrucciones, prestando atención a cómo modifican el registro F, prueba a debuggear el siguiente código paso a paso. Abre la carpeta *"ejercicios_flags/src"* y colócalo en **main.asm**, después de `EntryPoint`:

```
; Ejercicio 1.1 - Demostración de flags

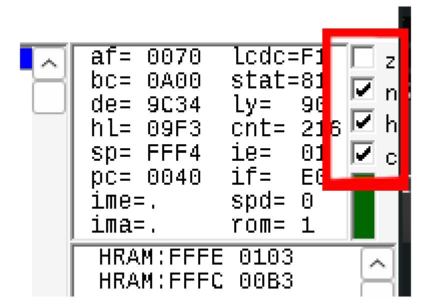
ld a, 0
ld b, 3
add a, b

; En este punto todos los flags estan a 0

dec a ; Aqui se activa n
dec a
dec a ; Aqui se activa z

jr @
```

Ahora quiero que te fijes en la esquina superior derecha del **debugger de BGB** podemos ver la representación de los distintos flags que contiene el registro F:



Sabremos si se han activado tras la última operación si tienen un check, es decir, si su valor es 1. Si quieres ver unas cuantas instrucciones y saber cuáles alteran el registro F te recomiendo ver **el tablón de operaciones** que hay en la plaza, justo aquí al lado.

➢ [**Si quieres ver el tablón de operaciones, pasa a la página 143**]

El Flag Z (Cero)



Flaguelito:

¡Madre mía que chapa te acabo de soltar! Cada día me parezco más a mi abuelo Flagbio con sus batallitas en el Reino de Bitrule. Aparentemente sus jarrones son los más famosos.

Bueno, bueno, que me voy por las ramas. Afortunadamente para tí para jugar en mi garito **tan solo vas a necesitar la ayuda del flag Z**. Mi hijo **Zerónimo** te lo puede explicar en un momentito.



Zerónimo:

¡Hola, encantado! ¿Te ha estado dando mucho la chapa mi padre? no te preocupes que esto va a ser rapidito.

Como ya sabrás, **el flag Z se activa únicamente si la última operación ha dado cero**. ¡Pues nuestros jarrones funcionan igual! Solo se rompen cuando el flag Z se activa.

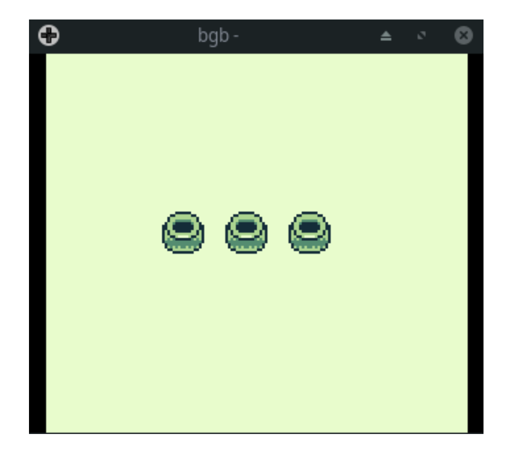
Rompe los jarrones



Zerónimo:

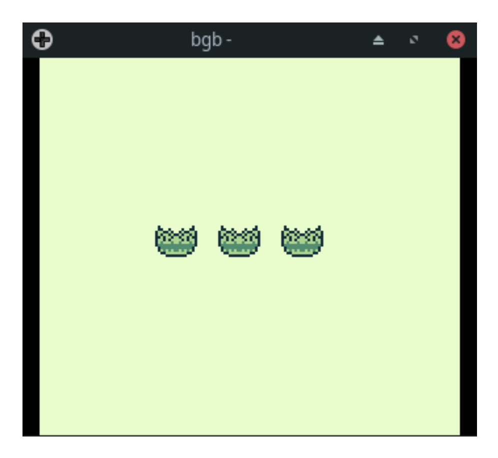
Ahora que ya sabes como funcionan nuestros jarrones, podemos pasar a la acción.

Abre el proyecto "*ejercicios_flags*". Si cargas **game.gb** en el emulador, deberías ver tres jarrones.



Tu objetivo es romper estos tres jarrones activando la flag Z tres veces.

Si lo haces correctamente deberías ver lo siguiente:



Para comenzar el ejercicio, abre el archivo "*ejercicios_flags/src/main.asm*".

¡Avísame cuando lo hayas conseguido!

Final de capítulo



Flaguelito:

¡Enhorabuena, bitsitante! Sabía que lo conseguirías.

Toma, quédate esto como recuerdo:



¡Has conseguido la ficha de Flaguelito!

¿Sabías que este flag también se suele usar para dar **saltos condicionales**?

¿Cómo? ¿Que no sabes que es un **condicional**?

¡No te apures joven! Justamente aquí al lado está **La noria de Loopita**. Sin duda ella te podrá explicar todo lo que necesitas sobre ellos.

¡Dale un saludo de mi parte!

Has completado el garito de Flaguelito. ¡Enhorabuena! ¿Qué quieres hacer ahora?

- **[Para continuar a la noria de Loopita, pasa a la página 109]**
- **[Para hablar con el resto de hijos de Flaguelito, pasa a la página 106]**
- **[Para volver a Pueblo Bandera, pasa a la página 97]**

Los hijos de Flaguelito

El flag N (Substracción)



Restario:

El flag N se utiliza para indicar si la última operación fue una resta.

- Si acabas de ejecutar una instrucción que contenga una resta (sub, dec, cp...) **el flag N se pone a 1.**
- Si has ejecutado cualquier otra instrucción, **el flag N se pone a 0.**

```
ld a, 5
sub 2    ; Resta -> el flag N se pone a 1
inc a    ; Incrementar el registro a en 1 -> el flag N se pone a 0
```

Aunque no se usa directamente para tomar decisiones (como el flag Z o el flag C), el flag N es importante internamente porque:

- Afecta cómo otras instrucciones interpretan el estado de la CPU.
- Es necesario para algunas operaciones combinadas que involucran aritmética y condiciones de salto.

Por ejemplo, algunas instrucciones de **DDA** (Decimal Adjust for Addition) dependen de saber si la operación previa fue suma o resta para corregir adecuadamente los valores **BCD** (Binary Coded Decimal).

El flag C (Acarreo)



Carryna:

El flag C se activa para indicar varias situaciones:

- Cuando el resultado de una suma de 8 bits sea mayor que `$FF`.
- Cuando el resultado de una suma de 16 bits sea mayor que `$FFFF`.
- Cuando el resultado de una resta o comparación sea menor que cero.
- Cuando una operación de rotación de bits desplace un bit a "1".

Por ponerte un ejemplo, si quieres sumar más caramelos de los que caben en una bolsa, necesitas otra bolsa. Eso es el acarreo (C = 1).

```
ld a, $F0   ; a = 240
add a, $20  ; a = 240 + 32 = 272 -> No cabe en 8 bits (se desborda)
; El resultado en el registro a será 272 mod 256 = 16
; El flag C se pone a 1 porque hubo acarreo.
```

Y si tus amigos te piden más caramelos de los que tienes, te quedarás en números negativos. Así que te tocará pedir un préstamo (C = 1).

```
ld a, $10   ; a = 16
sub $20     ; 16 - 32 -> no se puede (negativo)
; El resultado en el registro a bits será -16 mod 256 = 240
; El flag C se pone a 1 porque hubo un "borrow" (un préstamo).
```

El flag H (Medio acarreo)

Halfisto:
El flag H se activa para indicar si en la última operación ha habido acarreo desde los cuatro bits más bajos a los cuatro bits más altos en una operación aritmética que contenga sumas o restas.

- **Half-Carry:** En una suma, el flag H se pone a 1 si la suma de los 4 bits bajos se desborda (pasa de 15).

```
ld a, $0F   ; 00001111 (15 en decimal)
add a, $01  ; 00000001 (1 en decimal)

; La suma de los 4 bits bajos (1111 + 0001) = 10000 (16, desbordó a 5 bits).
; ¡Hubo acarreo en el medio! -> El flag H se pone a 1.
```

- **Half-Borrow:** En una resta, el flag H se pone a 1 si hubo un préstamo entre el bit 4 y el bit 3.

```
ld a, $10    ; 00010000 (16 en decimal)
sub $01      ; 00000001 (1 en decimal)

; (00010000 - 00000001) = 00001111 (15)
; 0 no puede restar 1 sin pedir prestado del siguiente bit más grande. Así que
aquí hay un préstamo entre el bit 3 y 4 -> El flag H se pone a 1.
```

El flag H existe principalmente para ayudar con operaciones **BCD** (Binary Coded Decimal). La CPU de la Game Boy puede hacer ajustes decimales usando la instrucción **DAA** (Decimal Adjust Accumulator), que necesita saber si hubo un half-carry o un half-borrow.

La noria de Loopita



Loopita:

¡Bitsitante, muchas gracias por venir!

¿A mi noria deseas subir?

¡Faltaría más!

Pero he de decirte,

esta noria no es como las demás.

Esta noria no es la clásica noria a la que te subes a dar una simple vuelta y se acabó lo que se daba.

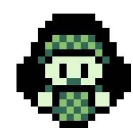
Tengo que reconocer que antiguamente sí que era así. Pero uno de estos días, frustrada porque no sabía cómo optimizar mi noria, mientras descansaba en el campo de girasoles estaba viendo los molinetes de colores girar y girar... y entonces se me iluminó la bombilla.

¡Pues claro! ¿Por qué no dejar que la gente elija cuantas vueltas quiere dar? ¿Y por qué no dejarles elegir la velocidad también? Bueno.. tras unos cuantos incidentes retiré esa opción pero, ¡Elegir el número de vueltas todavía se puede!

Simplemente tienes que programar las vueltas aquí en mi terminal. ¿Cómo? ¿Que por qué no lo hago yo? ¡Muy fácil! Porque si no, ¿quién va a vigilar que tú lo estés haciendo bien? Además, mi hipoteca de *Animal Coding* no se va a pagar sola.

Así que, ¡déjame explicarte cómo se hace!

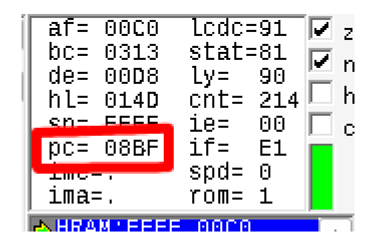
Saltos



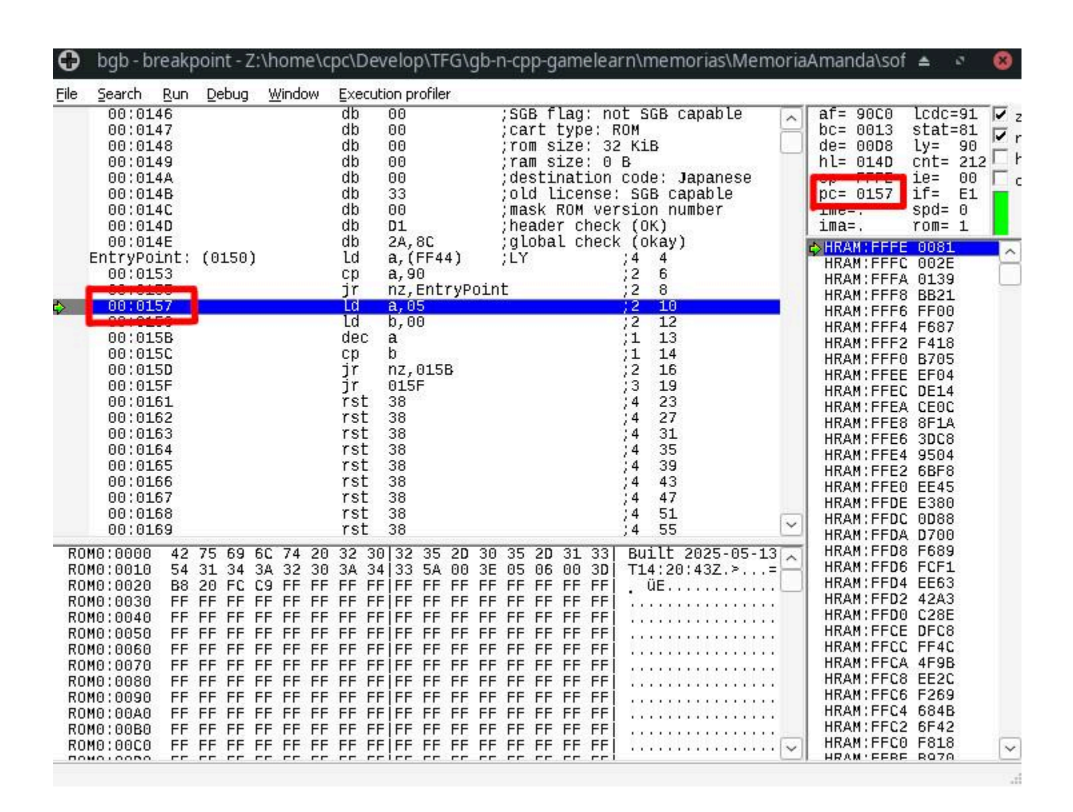
Loopita:

Antes de aprender lo que es un bucle, debes comprender lo que son **los saltos**.

El PC (Program Counter) es un registro de 16 bits que almacena direcciones de memoria. Es el que se encarga de indicar a la CPU la dirección de memoria dónde se encuentra la siguiente instrucción o dato que debe leer.



Puedes verlo en el **debugger de BGB**, en la esquina superior derecha:



Un salto viene a ser cuando modificamos el registro PC para indicarle qué parte de nuestro código queremos seguir ejecutando a continuación. Éste registro, al igual que el registro F, **es de sólo lectura**. Afortunadamente para nosotros, existen algunas instrucciones que nos permiten modificarlo para poder realizar estos saltos. Veamos un par de ellas:

Instrucción	Descripción
`jr` s8	Ejecuta un "salto relativo" de s8 (steps 8-bits) pasos en el PC (Program Counter). Los 8 bits son interpretados como bits con signo o signed bits, esto significa que esta instrucción solo puede saltar 128 bits hacia delante o hacia atrás (desde -128 hasta +127).
`jp` a16	Ejecuta un salto absoluto hacia la dirección de memoria a16 (address 16-bits) especificada.

Te preguntarás, ¿en qué situaciones es mejor usar `jr`? ¿por qué no usar la instrucción `jp` siempre y ya está?

Pues mira que te digo, **cada instrucción tiene sus ventajas y desventajas**. La instrucción `jr` es más lenta en ejecución que `jp`. Sin embargo, mientras que `jp` ocupa 3 bytes, `jr` ocupa 2 bytes, lo cual te permite ahorrar memoria en un entorno tan limitado como es la Game Boy, donde hasta el más mínimo byte puede resultar crítico. Si necesitáramos saltar 128 bytes o más, obligatoriamente usaríamos `jp`. Así que usar una instrucción u otra dependerá del contexto en el que te encuentres.

Veamos un par de ejemplos. Para empezar, con la instrucción `jr` hay que calcular cuántos bytes ocupan todas las instrucciones que hay en el camino hasta llegar a la que queremos saltar. En el caso de este ejemplo, saltaremos 3 bytes ya que la instrucción `jr` ocupa 2 bytes y `sub` ocupa 1 byte (2+1=3):

```
;La instrucción jr puede ir desde @-$80 (-128) hasta @+$7F (+127)
;el simbolo @ representa la dirección de la misma línea donde se encuentra

ld a, 5
ld b, 5

jr @+$03 ;la instrucción jr ocupa 2 bytes

sub b    ;sub ocupa 1 byte
inc a    ;aterrizaríamos aquí
```

Para `jp` podemos hacer lo mismo, aunque también podemos poner la dirección exacta donde queremos ir. Podemos calcular la dirección exacta o echarnos una mano del debugger para saber a qué dirección queremos ir:

```
ld a, 5
ld b, 5

jp $015F ;saltaríamos hasta inc a

sub b
inc a    ;$015F
```

¿Cómo vas? ¿Todavía no te has mareado verdad? ¡Genial!

Pues bien estos saltos son el primer paso para poder comprender cómo funcionan los bucles. Así que vamos a ver cómo se construyen a continuación.

Bucles



Loopita:

Voy a explicarte lo que son **los bucles**. Es como... ¡comer galletas!

Te dices a ti misma, venga Loopita te lo has ganado cómete una galleta. Así que te comes una. Sin embargo tu cerebro, que secretamente sabe programar y tiene otros planes para ti, te dice que te comas otra. Así que eso haces... hasta que te acabas el paquete, oops.

Un bucle se concibe como **realizar una misma acción un número determinado de veces** o hasta cumplir una condición, en mi caso... restar galletas hasta llegar a cero.

```
ld a, 100   ;A son las galletas en el paquete
ld b, 0     ;B son las galletas en mi tripa

dec a       ;decrementamos A
inc b       ;incrementamos B
jr @-$02    ;retrocede hasta la instrucción dec a

;¡Cuidado! Hemos creado un bucle infinito
```

Quiero que pongas especial atención a lo que te voy a decir ahora: **es MUY importante** que todos tus bucles tengan **siempre una condición de salida** claramente definida, ¡o la ejecución de tu programa se quedará eternamente atrapado en un **bucle infinito**! Y no queremos que la gente se quede dando vueltas en mi noria por toda la eternidad, sería injusto para los que hacen cola.

Te dirás Loopita, ¿y como hago para salir de éstos bucles? ¡Muy buena pregunta! Aquí es donde entran **los condicionales**, que nos ayudaran a poder controlar nuestros bucles.

Condicionales



Loopita:

Los condicionales son opciones extra que podemos añadir a ciertas instrucciones para tener un mayor control sobre el flujo de nuestro programa.

En nuestro caso, si los usamos en las instrucciones de salto `jp` o `jr` sirven para ejecutar dicho salto únicamente si la condición añadida es cumplida. Los condicionales que podemos usar son los siguientes:

Condicional	Descripción
z	La condición se cumple si el flag z está a 1 (la última operación dio 0 de resultado)
c	La condición se cumple si el flag c está a 1 (la última operación sobrepasó el límite de 8 bits del registro).
nz	La condición se cumple si el flag z está a 0.
nc	La condición se cumple si el flag c está a 0.

Y la sintaxis sería la siguiente:

```
jr (condicional), (dirección)
jp (condicional), (dirección)
```

Visto con un ejemplo quedaría así:

```
ld a, 5
ld b, 0

dec a           ;1 byte
cp b            ;1 byte
jr nz, @-$02    ;Si los registros A y B no son iguales (la instrucción cp
                ; no da 0), saltamos hacia la instrucción dec a
```

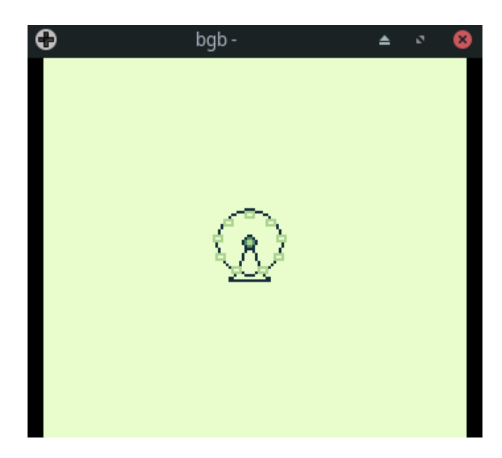
Las operaciones `jr` y `jp` son muy chulas, ¿verdad?
Pero hay que tener en cuenta que no todas las instrucciones permiten añadir condicionales.

Gira la noria



Loopita:

Abre el proyecto "*ejercicios_loops*". Si cargas **game.gb** en el emulador, deberías ver lo siguiente:



Tu objetivo es hacer girar la noria tantas veces como quieras usando saltos y condicionales.

Para comenzar el ejercicio, abre el archivo "*ejercicios_loops/src/main.asm*".

Te recomiendo echar un vistazo al tablón de operaciones para ver unas cuantas instrucciones y saber sus respectivos tamaños.

➢ [**Si quieres ver el tablón de operaciones, pasa a la página 143**]

Vuelve por aquí cuando te hayas divertido lo suficiente, ¡y cuidado con los bucles infinitos!

Final de capítulo

Loopita:

¡Bitsitante, muchas gracias por venir!
A esta noria siempre que quieras podrás subir,
¡Faltaría más!
Que el código te sea claro, que los bugs logres destruir,
¡y que cada salto condicional te haga sonreír!

Cof cof cof, ay perdona. Es que estar al lado de la carpa de **la adivina Madame Labelette** y escucharle hablar todo el día al final le come el coco a una.

¡Que pases un buen día, bitsitante!

….

¡Ah! Se me olvidaba, necesitarás esto:



¡Has conseguido la ficha de Loopita!

Has completado la noria de Loopita. ¡Enhorabuena! ¿Qué quieres hacer ahora?

- **[Para ver a la adivina Madame Labelette, pasa a la página 117]**
- **[Para volver a Pueblo Bandera, pasa a la página 97]**

La adivina Madame Labelette



Madame Labelette:

Vuestro porvenir puedo ver, un truco muy sencillo vais a aprender.

Si las direcciones de memoria no lográis memorizar
***las etiquetas** pueden ser una ayuda ejemplar*
ya que podréis señalar el lugar al que queréis saltar
¡dándole un nombre que si podáis recordar!

¿Acaso mis palabras han logrado prender fuego a vuestra curiosidad, oh bitsitante? ¡Loado sea! ¡Loado sea!

¿Quieres saber más sobre cómo las etiquetas van a mejorar tu futuro? Pues bien, prepárate una buena taza de té y presta atención a lo que te estoy a punto de contar.

La sintaxis de RGBASM nos permite identificar direcciones de memoria a través del uso de identificadores simbólicos, también llamados **etiquetas**. ¿Super conveniente, verdad? Así ya no tendrás que memorizar o calcular direcciones de memoria cada vez que quieras ejecutar un salto.

Pero antes de explicarte las etiquetas, vamos a ver qué son **los ámbitos**.

Ámbitos

Madame Labelette:

Un ámbito (o *scope*) es la región de un programa donde un elemento es válido, visible y accesible. Dicho elemento puede ser una variable, función o etiqueta entre otros elementos.

Imagínate que cada vez que entras a una feria (un ámbito), te dan una ficha especial. Dentro de esa feria, puedes gastar esa ficha en juegos o comida. Pero cuando vuelves a tu barrio (otro ámbito) e intentas usar esa ficha especial en el quiosco de la esquina, ésta no es válida. Es decir, sólo puedes usarla dentro del ámbito de la feria.

¡Así funcionan los ámbitos!

Existen varios tipos de ámbitos, entre ellos, **los ámbitos globales** y **los ámbitos locales**. Volvamos al ejemplo de la feria. Un ámbito global podría ser el recinto en sí, todo el mundo puede acceder a cualquiera de sus puestos y atracciones.

Dentro de ésta feria, cada puesto tiene su propio ticket de entrada y sus propias reglas. Éstos podrían ser ámbitos locales, ya que por ejemplo con un ticket para la montaña rusa no puedes acceder a la casa del terror. Al igual que tampoco puedes llevarte la pistola de tiro al blanco a los autos de choque...

En resumen, "*Feria*" es un ámbito global donde los diferentes puestos (noria, montaña rusa, etc) son ámbitos locales. En términos de programación se representarían como `ÁmbitoGlobal.ÁmbitoLocal`.

Es decir: `Feria.Noria`, `Feria.MontañaRusa`, etc...

Ahora, algo importante que deberías tener en cuenta es que **no puedes declarar varios elementos del mismo tipo con el mismo nombre** dentro de un mismo ámbito, o el pesado de **Mister Linker** se enfadará contigo y te responderá con un error de compilación.

Por ejemplo, dentro de *"Feria"* no pueden existir dos puestos de comida llamados *"El Ratón Glotón"*. Si le hablaras de uno de ellos a **Mister Linker**, no sabría a cual te estás refiriendo.

¡Pues todas éstas reglas también se aplican a las etiquetas!

Etiquetas



Madame Labelette:

¡Incluso existen diferentes tipos de etiquetas! Déjame explicarlas a continuación:

Etiquetas globales

Las etiquetas globales se pueden acceder **desde cualquier parte dentro del mismo fichero** en el que se haya declarado. Para declarar una etiqueta global solo basta con escribir el nombre que le quieras poner y añadir dos puntos (:) a continuación. ¿Sencillito, verdad?

pueblo.asm

```
ld a, 20    ; Tu dinero
ld b, 5     ; Precio de los tickets

Feria:                  ; Declaración de una etiqueta global
      sub b             ; Compra un ticket
      jr nz, Feria      ; Si tu dinero no es 0, comprar otro ticket

; Si ya no te queda dinero, sales de la feria

jr @
```

En éste ejemplo, sólo puedes acceder a la etiqueta global `Feria:` únicamente si estás referenciándola dentro de **pueblo.asm**.

Etiqueta local

Las etiquetas locales son aquellas que se pueden acceder **solo si estás dentro de su mismo ámbito**. Para definirlas basta con ponerles un nombre seguido de un puno (.). Se pueden anidar etiquetas locales dentro de etiquetas globales.

pueblo.asm

```
ld a, 20     ; Tu dinero
ld b, 5      ; Precio de los tickets

Feria:                       ; Declaración de una etiqueta global
      sub b            ; Compra un ticket
      ld c, 3                ; Vueltas que da el canguro loco

      .canguroLoco                 ; Declaración de una etiqueta local
            dec c                  ; Das una vuelta
            jr nz, .canguroLoco ; Si aún te quedan vueltas, dar otra

      jr nz, Feria      ; Si tu dinero no es 0, comprar otro ticket

; Si ya no te queda dinero, sales de la feria

jr @
```

En éste ejemplo, sólo puedes acceder a la etiqueta local `.canguroLoco` dentro del ámbito de la etiqueta global `Feria:`, que a su vez sólo puede usarse dentro de **pueblo.asm**.

Etiqueta exportada

Por último, tenemos las **etiquetas exportadas.** Se puede acceder a ellas **desde cualquier otro fichero.** Para definirlas hay que ponerles un nombre seguido de dos puntos dobles (::).

pueblo.asm

```
ld a, 20    ; Tu dinero
ld b, 5     ; Precio de los tickets

Feria:                        ; Declaración de una etiqueta global
      sub b             ; Compra un ticket
      ld c, 3                 ; Vueltas que da el canguro loco

      .canguroLoco                  ; Declaración de una etiqueta local
            dec c                   ; Das una vuelta
            jr nz, .canguroLoco ; Si aún te quedan vueltas, dar otra

      jr nz, Feria      ; Si tu dinero no es 0, comprar otro ticket

; Si ya no te queda dinero, sales de la feria

jp IrACasa
```

coche.asm

```
IrACasa::         ; Declaración de una etiqueta exportada
      ld a, 20    ; Distancia hasta casa

      .conducir                     ; Declaración de una etiqueta local
            dec a                         ; Restamos distancia
            jr nz, .conducir  ; Si la distancia no es cero, seguir
conduciendo

; Si la distancia es cero, ya hemos llegado

jr @
```

En éste ejemplo, puedes acceder a la etiqueta `IrACasa::`, declarada en **coche.asm**, desde el fichero **pueblo.asm** debido a que la hemos declarado como una etiqueta exportada.

Final de capitulo

Madame Labelette:

Ahora que ya conoces las etiquetas, ¿por qué no repasas tu código y lo limpias un poquito? Seguro que te ayuda a organizar todo mucho mejor y a no perderte en tu propio código.

Toma esto, te lo has ganado por haber aguantado mi eterno monólogo:



¡Has conseguido la ficha de Madame Labelette!

¡Que tengas un futuro próspero, bitsitante!

Has completado la adivina Madame Labelette. ¡Enhorabuena! ¿Qué quieres hacer ahora?

- **[Para continuar al circo de Funcelmo, pasa a la página 123]**
- **[Para volver a Pueblo Bandera, pasa a la página 97]**

El circo de Funcelmo

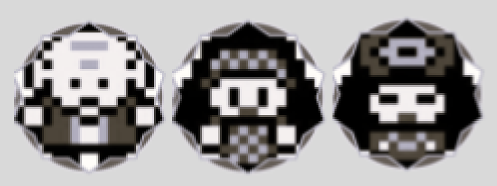
Taquilla



Taquillero:

Muy buenas bitsitante, ¿vienes a ver el espectáculo?

Pasar al siguiente apartado requiere los siguientes logros:



➢ **[Si ya has obtenido todos estos logros, pasa a la página 124]**



Taquillero:

Lo siento pero no te puedo dejar pasar si no tienes todas las fichas. **Te recomiendo pasar por el resto de garitos** antes de venir aquí.

Parece que todavía no puedes pasar por aquí. ¿Qué quieres hacer?

➢ **[Para volver a Pueblo Bandera, pasa a la página 97]**

Dentro del circo



????:

¡Hola a todo el mundo! Soy vuestro querido maestro de ceremonias, ¡**Funcelmo**!

Muchas gracias por haber venido a mi espectáculo. Como es costumbre, elegiré a alguien del público como voluntario para ejecutar mi función de hoy.

A ver, a ver... ¡Ah! ¿Qué ven mis ojos? ¡Pero si es el bitsitante!

¡Hoy serás tú quien salga al escenario! ¡Ven aquí, que no te de vergüenza!

Funciones



Funcelmo:

Así como un maestro de ceremonias organiza todo en un espectáculo, **una función** es un conjunto modular de instrucciones que organiza y ejecuta tareas específicas.

¡Cada función es como una actuación programada!

La ventaja de usar funciones en nuestro código es que **pueden ser llamadas desde otras partes de nuestro programa**, lo que permite evitar código duplicado cada vez que queramos ejecutar una misma acción.

Para crear una función básica en **RGBASM** necesitamos dos elementos:

- **Una dirección de memoria** donde empieza nuestra función, la cual podemos marcar usando una etiqueta.
- **Una instrucción** `ret` para marcar el final de nuestra función.

Las instrucciones call y ret

La instrucción `call` se usa para saltar a una dirección de memoria donde tienes una función.

A diferencia de `jp`, la instrucción `call` **guarda la dirección de memoria actual en la pila** (registro **SP**) para que más tarde puedas volver a esa dirección usando la instrucción `ret`.

La instrucción `ret` (return) extrae de la pila la dirección de memoria que se guardó al hacer `call`, lo que significa **volver al punto del programa de donde llamaste a esa función**.

Así puedes pausar lo que estabas haciendo, hacer otra tarea, y luego volver como si nada.

Tanto la instrucción `call` como la instrucción `ret` **permiten el uso de condicionales**.

Instrucción	Condicional	Dirección
CALL	(Z,C,NZ,NC)	d16
RET	(Z,C,NZ,NC)	x

Para llamar a nuestras funciones con la instrucción `call` necesitamos saber la dirección de memoria en la que se encuentran, para ésto usaremos una etiqueta.



Consejo del profesor:

"Añadir un comentario antes de cada función indicando que registros utiliza como parámetros y cuales modifica nos ayudará a leer rápidamente qué parámetros necesita cada vez que vayamos a invocarla sin necesidad de revisar el código."

```
; parametros(pasos -> b), modifica(a)
CuentaPasos:
      ld a, 0
      .loop
            inc a
            cp b
            jp nz, .loop
ret ; Volvemos al a misma dirección de donde fue llamada la función CuentaPasos

EntryPoint::

ld b, 10 ; Pasos a dar
call CuentaPasos ; Llamamos a la función que hemos declarado mas arriba

jr @
```

Las instrucciones PUSH y POP



Funcelmo:

Antes del grand finale, ¡llegó la hora del intermedio!

Mientras nos tomamos un respiro, démosle un fuerte aplauso a nuestros queridos e inseparables hermanos: ¡los payasos **Patapush** y **Patapop**!



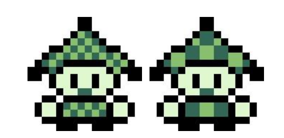
Patapush:

!Hola, hola! Yo soy Patapush.



Patapop:

¡Y yo soy Patapop!



Patapush y Patapop:

¡Y juntos somos los inseparables hermanos Patapush y Patapop!



Patapush:

¡Mirad cómo guardo un pañuelo en este sombrero mágico!

`PUSH pañuelo`

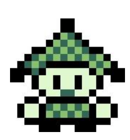


Patapop:

¡Y ahora yo, el asombroso Patapop, haré que ese pañuelo vuelva como por arte de magia!

`POP pañuelo`

¡Y voilà! El pañuelo está de vuelta en su sitio exacto... ¡sin arrugarse!

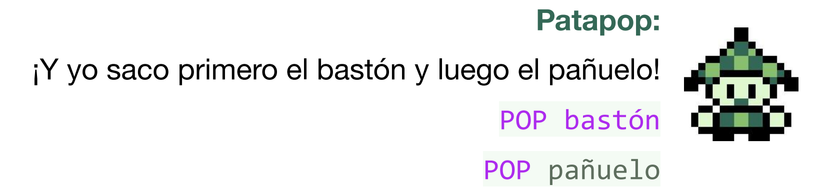


Patapush:

Todo lo que yo meta en el sombrero, Patapop lo puede sacar… ¡Pero ojo! ¡En orden inverso!

Primero meto el pañuelo, luego el bastón...

```
PUSH pañuelo
PUSH bastón
```

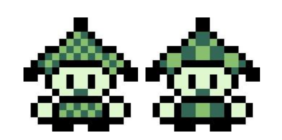


Patapop:

¡Y yo saco primero el bastón y luego el pañuelo!

```
POP bastón
POP pañuelo
```

¡Así funciona la pila, último en entrar, primero en salir!



Patapush y Patapop:

Si te has visto alguna vez en la situación de querer llamar a una función que modifica unos registros que contienen datos importantes y no deseas perder, no te apures. Aquí están las instrucciones `push` y `pop` para ayudarte.



Patapush:

La instrucción `push` **guarda** el contenido del registro de 16 bits que tu elijas (**HL**, **BC**, **DE** o **AF**) en la pila, de la siguiente manera:

1. El procesador reduce el puntero de pila (**SP**) en 2 (porque guarda 2 bytes).
2. Luego escribe el contenido del registro en esa nueva dirección de la pila.

Para **volver a recuperar** esos datos debemos usar **la instrucción** `pop`, que internamente hace lo siguiente:

1. Lee 2 bytes de dónde apunta el puntero de pila (**SP**).
2. Carga esos 2 bytes en el registro que le indiques.
3. Incrementa **SP** en 2 (porque ya "se sacaron" esos dos bytes).

Instrucciones PUSH	Instrucciones POP
`push bc`	`pop bc`
`push de`	`pop de`
`push hl`	`pop hl`
`push af`	`pop af`

Patapop:

Es muy importante que siempre que escribas una instrucción `push` para un registro, asegurarte de escribir un `pop` para el mismo. Así evitarás malinterpretar los datos y que **tu programa se descontrole**. Además, si haces muchos `push` sin hacer sus respectivos `pop`, la pila puede llenarse y pisar otros datos.

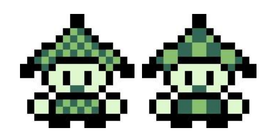


Patapush:

También hay que tener en cuenta que **el sistema de almacenamiento de la pila es LIFO** (Last In, First Out).

Ésto quiere decir que la pila es como una torre, el último valor del que hiciste `push`, tiene que ser es el primer valor del que haces `pop`. Es decir, que hay que seguir un orden específico a la hora de escribir estas dos instrucciones.

```
push bc
push de
      ; Tu código va aquí
pop de
pop bc
```



Patapush y Patapop:

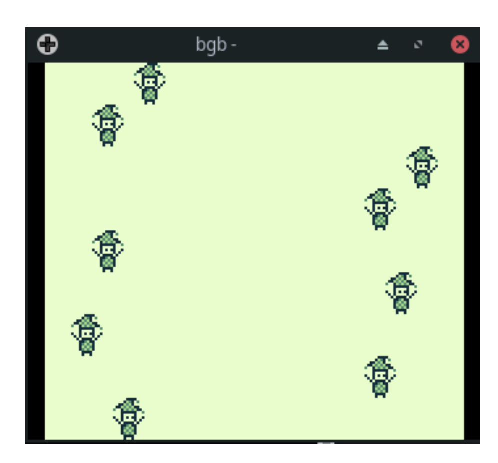
¡Y hasta aquí el intermedio! ¡Disfruten del resto del espectáculo!

Prueba final

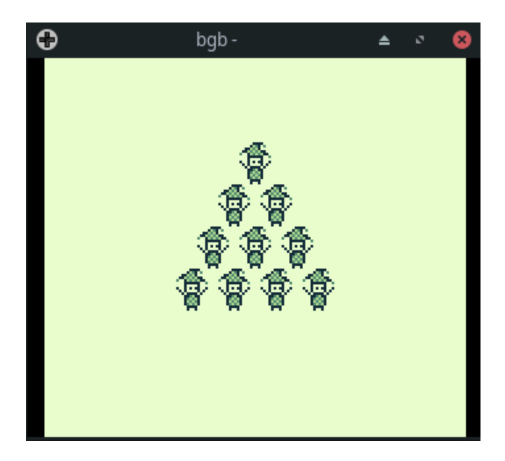


Funcelmo:

Abre el proyecto "*ejercicios_funciones*", Si cargas **game.gb** en el emulador, deberías ver a nuestros payasos equilibristas dándose un descanso:



Tu objetivo es ordenarles hacer una torre de payasos. Para ello usa todo lo aprendido en éste capítulo. Cuando lo consigas deberías ver algo así:



Para comenzar el reto, abre el archivo *"ejercicios_funciones/src/main.asm"*.

¡Buena suerte!

Final de capítulo

Funcelmo:

Es una torre de payasos magnífica ¡Un aplauso para el bitsitante!

Y con esto, señoras y señores, la función ha finalizado. Esperamos que lo hayan disfrutado tanto como nosotros. ¡Nos volveremos a ver!

[Y con esto, Funcelmo desaparece del escenario en una nube de humo. Parece que ha dejado algo atrás, tiene una etiqueta con tu nombre.]



¡Has conseguido la medalla de Funcelmo!

Has completado el circo de Funcelmo. ¡Enhorabuena! ¿Qué quieres hacer ahora?

➢ **[Para volver a Pueblo Bandera, pasa a la página 97]**

La gruta paleontológica

Ring, ring... Ring ring...

[Parece que el profesor Retromán te está haciendo una videollamada]

Prof. Retromán:

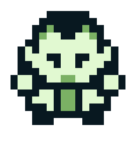
¿Cómo está yendo tu aventura?¿Cómo dices, que ahora mismo estás en una cueva muy extraña? Déjame ver... ¡ah! ¡Pero qué ven mis ojos, pero si es un yacimiento paleonto**lógico**!

¡Pero qué interesante!

¿Ves esas pintadas que hay en la pared de la cueva? Son unas enseñanzas ancestrales precursoras de la programación actual, **las operaciones lógicas**. Te pueden venir requetebién para saber hacer tus propias **máscaras de bits**, que son muy útiles para filtrar y extraer bits específicos de un registro.

¿Quieres que te explique cuales son **las operaciones lógicas** necesarias para saber hacer tus propias máscaras de bits?

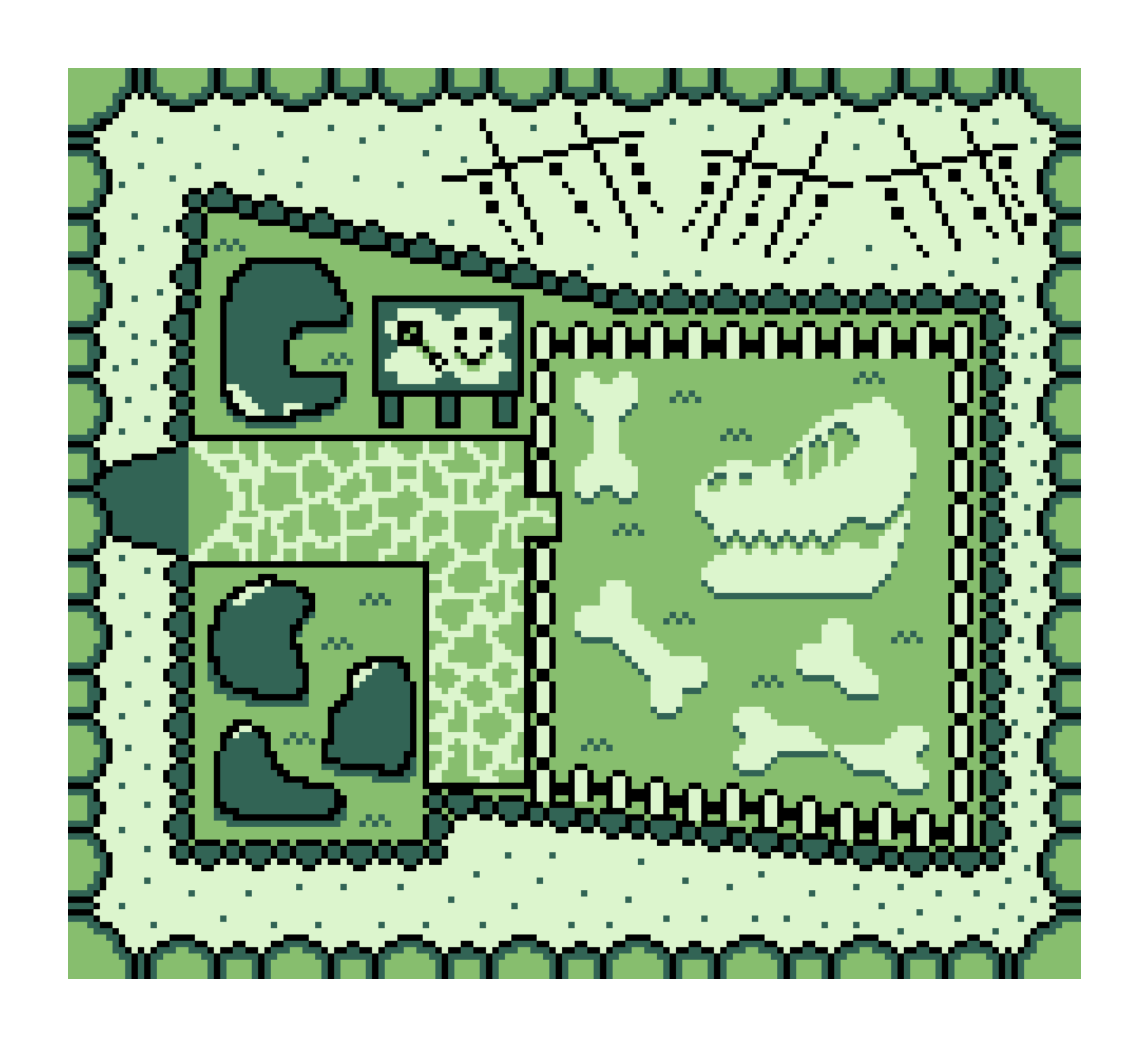
➢ **[Si quieres ver las operaciones lógicas, pasa a la página 136]**

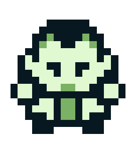


Prof. Retromán:

¡De acuerdo! ¿Y las **máscaras de bits**, quieres que te explique qué son y cómo se usan?

➢ **[Si quieres ver las máscaras de bits, pasa a la página 140]**





Prof. Retromán:
Muy bien, ya veo que estás haciendo un progreso excelente en tu aventura. ¡Sigue así!

Cling.

[El profesor ha colgado.]

➢ **[Para volver a Pueblo Bandera, pasa a la página 97]**

Operaciones lógicas

Prof. Retromán:

Para comenzar voy a explicarte lo que son **las operaciones lógicas**. Provienen de la lógica binaria, que trabaja únicamente con variables que solo pueden tomar dos valores discretos: Verdadero (1) o Falso (0). Son la base de los sistemas digitales y por ende la base de la estructura de los computadores.

De entre todas las operaciones lógicas que existen, las que más nos interesan son las operaciones `AND`, `OR` y `XOR`.

Operación lógica AND

Prof. Retromán:

La operación `AND` es una operación lógica que produce un resultado verdadero **únicamente cuando ambos operandos son verdaderos**. Puede compararse con una multiplicación, ya que solo da como resultado 1 si los dos valores son 1.

AND		
Operando 1	*Operando 2*	*Resultado*
0	0	0
0	1	0
1	0	0
1	1	1

Esta operación se usa para **enmascarar bits**, es decir, extraer solo los bits que te interesan de un resultado.

```
11111001
AND
00100011     ;Extraemos los bits de las posiciones 0, 1 y 5
-----------
00100001
```

Operación lógica OR



Prof. Retromán:

La operación `OR` es una operación lógica que resulta verdadera **cuando al menos uno de los operandos es verdadero**. Su comportamiento es similar al de una suma, aunque limitada al contexto binario (0+1=1, 1+1=1).

OR		
Operando 1	*Operando 2*	*Resultado*
0	0	0
0	1	1
1	0	1
1	1	1

Esta operación es útil para **activar bits** específicos sin afectar a los demás.

```
00001101
OR
00000010     ;Activamos el bit de la posición 1
-----------
00001111
```

Operación lógica XOR



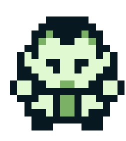
Prof. Retromán:

La operación `XOR` (o “OR exclusiva”) representa la función lógica de desigualdad, ya que devuelve un resultado verdadero **únicamente cuando ambos operandos son diferentes**. Si ambos valores son iguales, el resultado es falso.

XOR		
Operando 1	*Operando 2*	*Resultado*
0	0	0
0	1	1
1	0	1
1	1	0

Esta operación se suele usar para **alternar entre valores** o **detectar diferencias**.

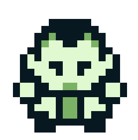
```
00001100
XOR
00001010
-----------
00000110     ;Los bits de las posiciones 1 y 2 tienen valores distintos
```



Prof. Retromán:

Ahora que conoces estas tres operaciones lógicas, ¿quieres que te enseñe lo que es una **máscara de bits** y sus aplicaciones?

➢ **[Si quieres ver las máscaras de bits, pasa a la página 140]**



Prof. Retromán:

Muy bien, ya veo que estás haciendo un progreso excelente en tu aventura. ¡Sigue así!

Cling.

[El profesor ha colgado.]

➢ **[Para volver a Pueblo Bandera, pasa a la página 97]**

Máscaras de bits



Prof. Retromán:

Las máscaras de bits son una técnica especialmente útil cuando se trabaja directamente con registros de memoria, como es el caso de la programación para Game Boy usando ensamblador.

Una máscara de bits es simplemente un valor binario que se usa junto con las operaciones lógicas `AND`, `OR` y `XOR` **para aislar o modificar bits** específicos de otro valor binario.

Piensa en ella como un colador que solo deja pasar los bits que te interesan, tapando los demás.

Vamos a verlo con un ejemplo, supongamos que tenemos el siguiente valor en el registro **A**:

```
ld a, $F0   ;A = $F0 = 11110000 en binario
```

Y quisiéramos activar el bit 2:

```
            ;Máscara para activar el bit 2 -> 00000100 = $04
or a, $04   ;11110000 OR 00000100 -> 11110100 = $F4
```

Ahora, si queremos borrar ese mismo bit:

```
            ;Máscara para desactivar el bit 2 -> 11111011 = $FB
and a, $FB  ;11110100 AND 11111011 -> 11110000 = $F0
```

Y por último, si quisiéramos invertir su valor:

```
            ;Máscara para invertir el bit 2 -> 00000100 = $04
xor a, $04  ;11110000 XOR 00000100 -> 11110100 = $F4
```

Registro LCDC



Prof. Retromán:

Las máscaras de bits son muy útiles a la hora de dibujar cosas en pantalla. Existe un registro donde se guarda el estado de la pantalla llamado **LCDC** (LCD Control). Éste registro se encuentra en la dirección `$FF40`. Cada uno de sus bits **activa diferentes opciones para el dibujado**:

Bit	Nombre	Descripción
7	LCD Control Operation	Enciende/apaga la pantalla
6	Window Tile Map Display Select	Selecciona mapa de tiles para la ventana
5	Window Display	Muestra u oculta la ventana
4	BG & Window Tile Data Select	Selecciona área de datos de tiles
3	BG Tile Map Display Select	Selecciona mapa de tiles para el fondo
2	OBJ Construction	Selecciona tamaño de sprites: 8x8 o 8x16
1	OBJ Display	Muestra u oculta los sprites
0	BG Display	Muestra u oculta el fondo

Un ejemplo de uso junto con una máscara de bits podría ser el siguiente:

```
ld a, %10000001   ; Enciende la pantalla y muestra el fondo
ld [$FF40], a
```

Si todavía no te has pasado por **Pueblo Palette**, ahora puede ser un buen momento. ¡Son todo unos expertos en el tema de dibujado!

Cling.

[El profesor ha colgado.]

Has llegado al final de la gruta. ¿Qué quieres hacer ahora?

- **[Para ir a Pueblo Palette, pasa a la página 57]**
- **[Para volver a Pueblo Bandera, pasa a la página 97]**

Tablón de operaciones

[Decides echar un vistazo al tablón de operaciones. Puedes leer lo siguiente]

Leyenda:	
`A`	Registro A
`r8`	Cualquier registro de 8 bits
`r16`	Cualquier registro de 16 bits
`s8`	Un paso de 8 bits (-128 hasta +127)
`a16`	Una dirección de 16 bits

Instrucción	Tamaño	Flags	Descripción
ADD `A`, `d8`	1 Byte	-	Suma A y d8, guarda el resultado en A
SUB `r8`	1 Byte	z	Le resta d8 a A, guarda el resultado en A
INC `r8`	1 Byte	-	Incrementa en 1 los contenidos del registro r8
DEC `r8`	1 Byte	-	Decrementa en 1 los contenidos del registro r8
JR `s8`	2 Bytes	-	Ejecuta un salto relativo a la dirección indicada.
JP `a16`	3 Bytes	-	Ejecuta un salto absoluto a la dirección indicada.
CP `r8`	1 Byte	z	Le resta r8 a A, actualizando los flags, pero no guarda el resultado

AND `r8`	1 Byte	z	Realiza la operación A and r8 y guarda el resultado en A
OR `r8`	1 Byte	z	Realiza la operación A or r8 y guarda el resultado en A
XOR `r8`	1 Byte	z	Realiza la operación A xor r8 y guarda el resultado en A

Has terminado de leer el tablón. ¿Qué quieres hacer ahora?

➢ **[Para volver a Pueblo Bandera, pasa a la página 97]**

Anexo



Edificio PPU Incertidumbre 21

`Suena un shubidubidubidowndown`

En un arrebato de curiosidad, te acercas a la calle Incertidumbre, al portal 21, justo como te indicó Paquito. Delante de ti ves un edificio de unas 5 plantas, con una fachada decimonónica que parece recién pintada. A pie de calle, ves un local con la persiana medio bajada. *Toc, toc, toc*.

Paco, el de la VRAM



¡Hombre! Ya pensaba que no vendrías. Así que quieres que te cuente desde mi perspectiva de director de cine de éxito, mi opinión larga y tendida sobre *Star Wars*. Sin ningún problema. Ah... ¿No era eso? ¿Que te siga contando cosas de la VRAM? ¿Con lo que te conté no te quedaste contento? Bueno, pues sigo...

La VRAM tiene unos pedazos de 8 KiB donde no te caben ni los créditos de una peli, pero bueno, todo sea dicho, te caben unos 384 *tiles*, donde cada *tile* pesa 16 bytes.

Zonas de la VRAM

Empieza en la dirección `$8000` y termina en la `$9FFF`. Pero no toda la memoria es igual; la PPU la interpreta de una determinada manera según la **zona**.

Es como una película de edición coleccionista con tomas falsas y comentarios del director; depende de qué parte del CD lea, el reproductor de vídeo te interpretará una cosa u otra. Bueno, quizás me esté flipando.

Te cuento lo de las zonas, que si no, me enrollo. Hay 2:

- De `$8000` a `$97FF` tenemos la zona dedicada a los *tiles*.
- De `$9800` a `$9FFF` tenemos la zona dedicada a los *tilemaps*.

Zona dedicada al tilemap

Esta se divide a su vez en 2 áreas de 32x32 *tilemaps* cada una:

- *Tilemap* 0: de `$9800` a `$9BFF`. Activado por defecto.
- *Tilemap* 1: de `$9C00` a `$9FFF`.

En cada *tilemap* se ponen los índices de 1 byte del *tile* a dibujar por pantalla en la dirección de memoria que estén puestos. Los *tiles* se obtienen de la zona dedicada a los *tiles*.

Podemos elegir qué *tilemap* queremos emplear para dibujar. Es como si tuviéramos 2 pantallas. Lo podemos hacer gracias al **registro LCDC**. Este registro nos indica en su 6.º bit si vamos a usar el ***tilemap* 0** (si bit 6 = **0**) o el ***tilemap* 1** (si bit 6 = **1**).

Para referirnos a este bit en concreto, le llamaremos *LCDC.6*.

Con el LCDC, también podemos **elegir qué bloque de *tiles*** queremos usar **para el fondo y la ventana**. En su 4.º bit podemos indicar qué modo de direccionamiento queremos usar; ahora te lo cuento.

Zona dedicada a los tiles

Esta se divide a su vez en **3 bloques**, de 128 *tiles* cada uno. Cada *tile* de esta zona tiene un **identificador** para poder referenciar en el *tilemap*.

Te dejo aquí los bloques y los identificadores:

- Bloque 0: de `$8000` a `$87FF`. *Tiles* de 0 a 127 (`$00 - $7F`). Aquí se encuentran los *tiles* de los **Objetos**.

- Bloque 1: de `$8800` a `$8FFF`. *Tiles* de 128 a 255 (`$80 - $FF`).
- Bloque 2: de `$9000` a `$97FF`. *Tiles* de 0 a 127 (`$00 - $7F`).

Como te comentaba antes, según LCDC.4:

Si LCDC.4	*Tiles* 0-127	*Tiles* 128-255	Método de direccionamiento
1	Bloque 0	Bloque 1	Método `$8000`: puntero base = `$8000`
0	Bloque 2	Bloque 1	Método `$8800`: puntero base = `$9000`

Bien, creo que no me queda nada para explicarte. ¡Vuelve cuando quieras una peli!

Al salir del videoclub, pasas por delante del portal del edificio, que tiene una alta puerta de madera. Te quedas mirando los telefonillos:

- `$0000`: Portería.
- `$8000`: Videoclub.
- `$C000`: Los PPuesta.
- `$E000`: Bitcenta, Mabitsa y Concha.
- `$FEA0`: Ascensor (prohibido).
- …

De repente te irrumpe una voz solemne te increpa.

Juan PPUesta



Me llena de orgullo y satisfacción darle la bienvenida a este, nuestro edificio. Yo soy Juan PPuesta, presidente de la comunidad. Aquí, como irás viendo, somos como una gran familia.

Ah... que no vienes a ver el piso que está de alquiler... Y... Entonces, ¿a qué ha venido usted? ¿Quería hablar conmigo? Pues no tengo nada que contar, fuera de mis deberes presidenciales. Que si quiere, yo se los cuento encantado, vaya.

A ver, primer punto del día: la **PPU**. La unidad de Procesamiento de Píxeles, PPU (*Pixel Processing Unit*), es la encargada de renderizar el juego por la pantalla de la Game Boy. Como un buen presidente, maneja el cotarro.

Modos PPU

En un *frame*, los ciclos de la PPU entran en **4 modos posibles**:

Modo	Descripción	VRAM
3	PPU dibuja (envía píxeles a la pantalla).	Inaccesible
2	La PPU busca *sprites* que intersequen con la siguiente línea.	Inaccesible menos para los OBJ
1	*Vertical Blank*, **VBlank**. La PPU "se espera" 10 líneas después de dibujar la pantalla entera mientras espera al siguiente *frame*.	Accesible
0	*Horizontal Blank*, **HBlank**. La PPU "se espera" al terminar de pintar una línea hasta la siguiente.	Accesible

Y tú puedes pensar "*¡Qué follón!*", pero es un ciclo bastante sencillo. Hace algo como "pico, mazo, pico, pico, mazo"... Más bien en 1 *frame*, **la PPU alterna entre 2 -> 3 -> 0** unas 144 veces **y luego** pasa al **modo 1**. Puede que te suene algo del scanline o similar...

Cabe destacar que **los modos son periodos de tiempo**, donde cada uno no dura lo mismo. La **VRAM** sólo es **accesible** en los **Modos 0 y 1**.

Registro STAT

Para ver en qué modo está la PPU, nos fijaremos en el **registro STAT** situado en `$FF41`. Es como una mirilla de unas que yo me sé. Quizás lo veas escrito también como rSTAT. Este registro tiene 8 bits y cada uno de ellos tiene un significado.

Bit	7 6 5 4 3 2	1 0
Qué controla	*irrelevante de momento*	**Modo PPU**

Estos **bits 0 y 1**, los menos significativos, indicarán qué modo estamos usando. Según la combinación de estos dos bits (en binario) podemos ver a qué modo corresponde:

Bit 1	Bit 0	Modo
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**3**

Viendo esto, podemos decir que si STAT tiene el **`bit 1 = 0`**, la VRAM estará disponible. Ya usarás esta información como consideres.

La PPU puede mostrar por pantalla elementos que están divididos en uno de los 3 tipos de capas que puede manejar la consola: fondo, *sprites* y ventana.

Espero que esta información haya sido de su agrado y vuelva muy pronto a visitarnos; si es para ver el piso, mejor.

Bibliografia

- Rhey T. Snodgrass and Victor F. Camp - Page 96 of Radio Receiving for Beginners. Rhey T. Snodgrass and Victor F. Camp (copyright 1922 by The MacMillan Company, New York). Recuperado de https://commons.wikimedia.org/wiki/File:International_Morse_code.png?uselang=es#Licencia